

AISB 2004 Convention:

Motion, Emotion and Cognition

AISB

The Society for the Study of Artificial
Intelligence and the Simulation of Behaviour

Proceedings of the AISB 2004

Fourth Symposium on Adaptive Agents and Multi-Agent Systems

Co-sponsored by



29 March – 1 April, 2004

ICSRiM, University of Leeds, Leeds LS2 9JT, UK

www.leeds.ac.uk/aisb

www.icsrim.org.uk

AISB 2004 Convention

29 March – 1 April, 2004

ICSRiM, University of Leeds, Leeds LS2 9JT, UK
www.leeds.ac.uk/aisb www.icsrim.org.uk

Proceedings of the AISB 2004 Fourth Symposium on Adaptive Agents and Multi-Agent Systems

Published by

**The Society for the Study of Artificial Intelligence and the
Simulation of Behaviour**

<http://www.aisb.org.uk>

ISBN 1 902956 35 8

Contents

The AISB 2004 Convention	iii
<i>K. Ng</i>	
Symposium Preface	iv
<i>D. Kudenko, E. Alonso, D. Kazakov</i>	
Keynote Lecture:	
Gödel Machines and other Wonders of the New, Rigorous, Universal AI	1
<i>J. Schmidhuber, IDSIA, Switzerland</i>	
Long Papers:	
The Role of Environment Structure in Multi-Agent Simulations of	2
Language Evolution	
<i>M. Bartlett, D. Kazakov</i>	
Baselines for Joint-Action Reinforcement Learning of Coordination in	10
Cooperative Multi-Agent Systems	
<i>M. Carpenter, D. Kudenko</i>	
Time Management Adaptability in Multi-Agent Systems	20
<i>A. Helleboogh, T. Holvoet, D. Weyns</i>	
Reinforcement Learning of Coordination in	30
Heterogeneous Cooperative Multi-Agent Systems	
<i>S. Kapetanakis, D. Kudenko</i>	
Evolving the Game of Life	37
<i>D. Kazakov, M. Sweet</i>	
Dynamic and Distributed Interaction Protocols	45
<i>J. McGinnis, D. Robertson</i>	
Robust Online Reputation Mechanism by Stochastic Approximation	55
<i>T. Sakai, K. Terada, T. Araragi</i>	
Multi-Agent Coordination in Tree Structured Multi-Stage Games	63
<i>K. Verbeeck, A. Nowe, M. Peeters</i>	
A Role Based Model for Adaptive Agents	75
<i>D. Weyns, K. Schelfhout, T. Holvoet, O. Glorieux</i>	
Dynamic Scheduling of Multiple Agents for a Heterogeneous Task Stream	86
<i>N. Windelinckx, M. Strens</i>	

Short Papers:

Interactive Multi-Agent Learning	95
<i>P.R. Graca, G. Gaspar</i>	
Using XCS to Build Adaptive Agents	101
<i>Z. Guessoum, L. Rejeb, O. Sigaud</i>	
The Strategic Control of an Ant-Based Routing System using Neural Net Based Q-Learning Agents	107
<i>D. Legge, P. Baxendale</i>	
A Motivation-based Approach for Autonomous Generation and Ranking of Goals in Artificial Agents	113
<i>L. Macedo, A. Cardoso</i>	
Self-adaptive Network of Cooperative Neuro-agents	119
<i>J.P. Mano, P. Glize</i>	
A Multi-Agent Framework for Web Information Retrieval	124
<i>P.A.C Sousa, J.P. Pimentão, B.R.D. Santos, A.S. Garção</i>	
Automated Component-Based Configuration: Promises and Fallacies	130
<i>S. van Splunter, N. Wijngaards, F. Brazier, D. Richards</i>	

The AISB 2004 Convention

On behalf of the local organising committee and all the AISB 2004 programme committees, I am delighted to welcome you to the AISB 2004 Convention of the Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB), at the University of Leeds, Leeds, UK.

The SSAISB is the oldest AI society in Europe and it has a long track record of supporting the UK AI research community. This year, the underlying convention theme for AISB 2004 is “*Motion, Emotion and Cognition*”, reflecting the current interest in such topics as: motion tracking, gesture interface, behaviours modelling, cognition, expression and emotion simulation and many others exciting AI related research topics. The Convention consists of a set of symposia and workshop running concurrently to present a wide range of novel ideas and cutting edge developments, together with the contribution of invited speakers:

- Prof Anthony Cohn
Cognitive Vision: integrating symbolic qualitative representations with computer vision;
- Prof Antonio Camurri
Expressive Gesture and Multimodal Interactive Systems;
- Dr David Randell
Reasoning about Perception, Space and Motion: a Cognitive Robotics Perspective; and
- Dr Ian Cross
The Social Mind and the Emergence of Musicality,

not to mention the many speakers invited to the individual symposia and workshop, who will made the Convention an exciting and fruitful event.

The AISB 2004 Convention consists of symposia on:

- Adaptive Agents and Multi-Agent Systems;
- Emotion, Cognition, and Affective Computing;
- Gesture Interfaces for Multimedia Systems;
- Immune System and Cognition;
- Language, Speech and Gesture for Expressive Characters; and the
- Workshop on Automated Reasoning.

The coverage is intended to be wide and inclusive all areas of Artificial Intelligence and Cognitive Science, including interdisciplinary domains such as VR simulation, expressive gesture, cognition, robotics, agents, autonomous, perception and sensory systems.

The organising committee is grateful to many people without whom this Convention would not be possible. Thanks to old and new friends, collaborators, institutions and organisations, who have supported the events. Thanks the Interdisciplinary Centre of Scientific Research in Music (ICSRiM), School of Computing and School of Music, University of Leeds, for their support in the event. Thanks to the symposium chairs and committees, and all members of the AISB Committee, particularly Geraint Wiggins and Simon Colton, for their hard work, support and cooperation. Thanks to all the authors of the contributed papers, including those which were regretfully not eventually accepted. Last but not least, thanks to all participants of AISB 2004. We look forward to seeing you soon.

Kia Ng

AISB 2004 Convention Chair
ICSRiM, University of Leeds,
School of Computing & School of Music,
Leeds LS2 9JT, UK
kia@kcng.org www.kcng.org

Proceedings of the AISB 2004 Fourth Symposium on Adaptive Agents and Multi-Agent Systems (AAMAS-4)

Symposium Preface

The AAMAS Symposium series has now reached its fourth year, and by now has become a well-established event. In fact, this fourth symposium is the biggest yet, with 17 papers to be presented. This clearly shows that the research area of adaptive and learning agents is active and is generating growing interest. Nevertheless, three years after the first AAMAS Symposium, there are many major challenges left to tackle. While our understanding of learning agents and multi-agent systems has advanced significantly, most evaluations are still on simple scaled-down domains, and, in fact, most methods do not scale up to the real world. This is a major obstacle to overcome in order to promote learning agent technology for commercial applications. Stay tuned for new developments.

We would like to thank Juergen Schmidhuber for agreeing to give this year's distinguished keynote talk. We also thank the members of the AAMAS-4 program committee for fast and thorough reviews. Last, but not least, our thanks to AgentLink-III for co-sponsoring the Symposium, to AISB for once again providing outstanding help in organization of this event, and of course special thanks to the authors without whose high-quality contributions there wouldn't be a Symposium to begin with.

Daniel Kudenko, Eduardo Alonso, Dimitar Kazakov

Symposium Chair: Daniel Kudenko, University of York, UK.

Co-Chairs: Eduardo Alonso, City University, UK.
Dimitar Kazakov, University of York, UK.

Program Committee:

Frances Brazier, Vrije Universiteit, Amsterdam, Netherlands.
Philippe De Wilde, Imperial College London, UK.
Kurt Driessens, Catholic University of Leuven, Belgium.
Saso Dzeroski, Jozef Stefan Institute, Slovenia.
Zahia Guessoum, LIP 6, Paris, France.
Tom Holvoet, Catholic University of Leuven, Belgium.
Paul Marrow, Btexact Technologies, UK.
Ann Nowé, Free University Brussels, Belgium.
Luis Nunes, University of Porto, Portugal.
Eugenio Oliveira, University of Porto, Portugal.
Paolo Petta, Austrian Research Centre for AI, Austria.
Enric Plaza, IIIA-CSIC, Spain.
Malcolm Strens, QinetiQ, UK.
Marco Wiering, University of Utrecht, Netherlands.
Niek Wijngaards, Vrije Universiteit, Amsterdam, Netherlands.

Keynote Lecture

Gödel Machines and other Wonders of the New, Rigorous, Universal AI

Jürgen Schmidhuber

IDSIA

Lugano, Sitzerland

juergen@idsia.ch

In arbitrary environments governed by unknown computable probabilistic laws (this includes partially observable environments and essentially everything we can write papers about), the ultimate predictor is Solomonoff's Bayesian induction scheme based on the universal prior M , the enumerable weighted sum of all enumerable measures. In theory we may use M online to predict consequences of future action sequences, always choosing actions with highest predicted success. This approach yields Hutter's (IDSIA) optimal general reinforcement learner AIXI, which we will briefly review.

The main problem is that M and AIXI are uncomputable. This leads to the next question: what is the best general reinforcement learner, given realistic, limited computational resources? The answer lies in the concept of proof search.

Usually work in machine learning proceeds like this: (1) invent a new learning algorithm, (2) prove its usefulness in a particular situation or set of situations, (3) apply, (4) publish. Normally step (2) is of interest only when the proof embodies some shortcut, that is, when we do not have to execute the entire algorithm in every possible scenario to figure out whether it is really good or not. Quickly producible proofs of some algorithm's quality may be viewed as efficient predictive models, given certain axioms describing what is known about environment and embedded agent. Proof systems are more general than most traditional predictors though. For example, proofs may make both low-level predictions and predictions limited to useful temporal / spatial abstractions that ignore irrelevant detail, etc.

Let us automatize steps (1-3) above in the most general possible way, such that any aspect of some reinforcement learner's behavior in some given environment is potentially subject to self-improvements, provided that the self-improvement's quality is provable at all through some proof of the shortcut type above. (If it is not, then there is no reason to believe that human - as opposed to automatic - insight might help.)

This leads us to the Gödel machine. It can be implemented on a traditional computer and solves arbitrary computational problems in an optimal fashion inspired by Kurt Gödel's celebrated self-referential formulas (1931). It starts with an axiomatic description of itself, and we may plug in any utility function, such as the expected future reward of a robot. Using an asymptotically optimal proof searcher which systematically tests proof-computing programs called proof techniques, the Gödel machine will rewrite any part of its software (including the proof searcher) as soon as it has found a proof that this will improve its future performance, given the utility function and the typically limited computational resources. Self-rewrites are globally optimal (no local minima!) since provably none of all the alternative rewrites and proofs (those that could be found by continuing the proof search) are worth waiting for.

To initialize the proof searcher we may use the recent Optimal Ordered Problem Solver OOPS. We will show results of various OOPS applications.

The Role of Environment Structure in Multi-Agent Simulations of Language Evolution

Mark Bartlett, Dimitar Kazakov*

* Department of Computer Science

University of York

Heslington, York YO10 5DD

bartlett@cs.york.ac.uk, kazakov@cs.york.ac.uk

Abstract

This paper presents a multi-agent system which has been developed in order to test our theories of language evolution. We propose that language evolution is an emergent behaviour, which is influenced by both genetic and social factors and show that a multi-agent approach is thus most suited to practical study of the salient issues. We present a hypothesis that the original function of language in humans was to share navigational information, and show experimental support for this hypothesis through results comparing the performance of agents in a series of environments. In particular, we study how the degree to which language use is beneficial varies with a particular property of the environment structure, that of the distance between resources needed for survival.

1 Introduction

The use of computer simulations to study the origin and evolution of language has become widespread in recent years (Briscoe, 2002; Kirby, 2002; Steels, 1999). The goal of these studies is to provide experimental support for theories developed by linguists, psychologists and philosophers regarding the questions of how and why language exists in the form we know it. As verbal language by its very nature leaves no historical physical remains, such as fossils, computer simulation is one of the best tools available to study this evolution, allowing us to construct a model which simulates the relevant aspects of the problem, and to abstract away any unnecessary detail. Experiments conducted using such an approach can be performed much more quickly than those involving teaching language to apes or children and allow researchers to study language in situations which would be impossible with live subjects.

Our multi-agent system has been designed to test the theory that language could potentially have evolved from the neural mechanisms our ancestors used to navigate (Kazakov and Bartlett, 2002; Hauser et al., 2002; O’Keefe and Nadel, 1978). Specifically we wish to explore the feasibility of the hypothesis that the original task of speech in humans was to inform others about the geography of the environment. To this end, we have constructed an artificial-life environment in which a society of learning agents uses speech to direct others to resources vital to survival, using the same underlying computational mechanism as they use to navigate. We discuss the altruistic nature of this activity and evaluate the benefits this brings to the community by measuring the difference in

performance between populations of agents able to communicate and those unable to do so. We also evaluate the performance of a population of agents engaging in a non-communicative act of altruism, namely sharing stores of resources. Our simulations are carried out in a series of environments in which the distance between the two resources needed for survival is varied to assess what impact this feature of the terrain may have on any benefit that language may bring. We also study how the distance between resources of the same type may affect performance. Our aim in this paper is to assess the conditions in which language use may be beneficial and hence could be selected for by evolution.

We have chosen to carry out our simulations within a multi-agent setting as the nature of these systems allows us easily to capture much of the behaviour we wish to program into our models. We assume that language has both innate components (such as the willingness to speak) and social components (such as the words used in a language). Multi-agent systems allow both these aspects to be modelled relatively simply. Though the genetic nature of language could be modelled equally well using only genetic algorithms, indeed some researchers have done this (Zuidema and Hogeweg, 2000; Oliphant, 1997), in order to model the social aspects of language which rely on phenotypical behaviour, situatedness, grounding and learning, it is necessary to employ an agent model.

Simulations of the evolution of language using the multi-agent paradigm can also be of interest to the designer of any general-purpose agent-based application. In a dynamic environment that is expected to change considerably during an agent’s lifetime, the faculty of learning could be essential to its success. In an evolutionary MAS

setting, sexual reproduction and mutation can be used to explore a range of possible learning biases, from which natural selection would choose the best. One would expect Darwinian evolution to advance in small steps and select only very general concepts. One could also implement Lamarckian evolution, that is, use a MAS in which the parent's individual experience can be reflected in the learning bias inherited by their offspring. Lamarckian evolution is faster but brings the risks of inheriting concepts that were relevant to the parents but are not reflected in the new agent's lifetime. This work explores what could be a third way of evolving a learning bias, through communication. Language uses concepts that are specific enough to be useful in the description of an agent's environment, yet general enough to correspond to shared experience. In this way, language serves as a bias inherited through social interaction rather than genes. In the current work, language serves as a way to share a bias for navigating in an environment.

2 Altruism and Sharing

Language use of the kind presented in this work is an inherently altruistic act; by informing another agent of the location of resources, an agent increases both the survival prospects of his rival and the probability that the resource mentioned will be exhausted before he is able to return to it. In our previous work (Turner and Kazakov, 2002) which focussed on resource sharing as a form of altruism, we found that through the mechanism of kin selection (Hamilton, 1964; Dawkins, 1982) it is possible to promote altruistic acts in an agent population if the degree of relatedness between the agents is known and the amount of resource given is able to be measured and controlled. It was also found that a poor choice of sharing function (which is used to decide the amount of resource shared) could lead to a population in which all agents had enough energy to survive, but insufficient to reproduce. This can lead to extinction of a large population even in the circumstances where a smaller population would be able to survive.

Quantifying the cost of the act is much harder to achieve when the altruistic act is to share knowledge of a resource location. In place of the one-off payments of resources that occur when resources are shared, informing an agent of the location of a resource can involve a long-term, sizeable drain on the resource, especially if the receiver goes on to inform others. Conversely, the act may result in no cost at all in the case where the receiver is in such need of the resource that it is unable to reach it before dying. This point also illustrates another problem with using language to share, it is also hard to estimate the benefits that the altruistic act will bring to the listener. When resources are given directly to another agent, the amount of help is easy to quantify, whereas when language is used the reward is delayed for several turns and

may never be achieved, as explained previously.

While altruism based on the underlying assumption of kin selection is used here due to the desire to provide a feasible explanation for a phenomenon in an artificial life setting, the idea of kin selection provides a useful metaphor for any community of agents. In a collection of agents solving a variety of tasks with limited resources, the degree to which two agents should cooperate (and share resources, if needed) should intuitively be related to the proportion of tasks they have in common: if we view the relatedness of two agents as the proportion of tasks they share, the ideas behind kin selection and neo-Darwinism provide inspiration for how agents might best cooperate to achieve their collective goals.

3 The Multi-Agent System

In order to simulate language evolution, we have created a multi-agent system. The York Multi-Agent System (Kazakov and Kudenko, 2001) is a Java based application which allows for artificial life simulations to be conducted in two dimensional environments. It is particularly well suited to studying learning and evolution. Agents in this system have a behaviour based on drives, the most important of which are hunger and thirst. At each time-step, after the values of the drives are increased to reflect the cost of living, an agent will attempt to reduce whichever of its drives is greater. If either drive reaches its maximum then the agent dies and is removed from the simulation. Agents can also die of old age. This can occur starting when the agent is 300 cycles old.

If two agents with sufficiently low hunger and thirst values share a location, they may mate to produce a third agent. Mating costs an agent a one-off payment of one third of its food and water reserves, with the amount subtracted from both parents going to the child as its initial levels.

Agents attempt to reduce their hunger or thirst by finding and using food and water resources respectively. A resource can be utilised by an agent if they share a square in the environment, and this decreases the appropriate drive to a minimum and reduces the amount of resource remaining at this location. When a resource is entirely depleted by use, it is removed for several turns after which it reappears at its previous location with its resources renewed. When resources are required, agents will first utilise any in the square they occupy or will look for resources in adjacent squares. If this fails, they attempt to generate a path to a known resource as will be explained below and, finally if all else fails, they will make a random exploratory move. Additionally, in some experiments performed, agents are able to request resources or directions from other agents. Sharing resources uses a progressive taxation policy with 50% of any resources in excess of the minimum needed for reproduction being given to the requesting agent, while directions are shared in the man-

ner which will be described below. Agents always share when they are asked for help (assuming that they are able to) and they will do so as if they were clones of a single genotype: there is no account taken of the degree of kinship when deciding how much help to give.

4 Representation of Knowledge

The information used to navigate and communicate about the location of resources is stored in the form of rules which contain ordered sequences of landmarks, or paths, which are to be passed while travelling from the current location to the target resource. The landmarks used for this purpose are items within the environment whose function as landmarks and whose names are assumed to be known by all agents throughout the whole simulation. Informing another agent of a resource through the exchange of a path is an altruistic act for the reasons discussed in previous sections.

Agents can acquire knowledge of new paths in 2 ways. Agents may obtain paths through linguistic methods as noted above or they may find resources through exploration, in which case they will be able to construct a path linking this new resource to the previous location they visited by recalling the landmarks they have seen on their journey. In either case, the agent will store the new path internally in the same data structure in a form equivalent to

$$goto(resource) \rightarrow goto(locX), l_1, l_2, l_3, \dots, l_i$$

in the case of a rule acquired through communication and in a form similar to the following for rules acquired through exploration.

$$goto(resource) \rightarrow goto(locY)$$

$$goto(locY) \rightarrow goto(locX), l_1, l_2, l_3, \dots, l_i$$

where l_1, \dots, l_i are the landmark names received, in the case of linguistic acquisition, or the landmarks seen during the journey, in the case of exploration, and $locX$ is the location the agent gained the knowledge, in the case of linguistic acquisition, or the previous location visited, in the case of exploration. $locY$ is the location of the resource itself in the latter example. $locX$ and $locY$ are stored as a list of landmarks visible from the location in question (range of sight is limited to 2 squares in all directions). The first rule given above can be understood as stating that to go to the resource is equivalent to first going to $locX$ and then passing the given landmarks in order. The second rule set can similarly be understood as stating that to go to the resource it is sufficient to go to $locY$ which can be achieved by going to $locX$ and then passing the landmarks in the stated order. Note that the path that can be generated to take an agent from $locX$ to the resource is the same using the either first rule or the rule set given above, the only difference being that in the latter case the agent

has already visited the actual resource and thus can store a description of the environment at that point. A further rule of similar form is added to an agent's knowledge base when new paths are acquired through exploration that enable paths to be traversed in reverse direction. It should be noted that this description is very impoverished, containing no information on factors such as direction, absolute position or distance (aside from that which can be inferred from the number of landmarks mentioned in the rule).

These rules can be viewed in two ways, and are used as such by the agents. Firstly, they can be seen as procedural rules which capture the spatial relationship between locations and can be used for navigation by agents to resources and secondly, as grammar rules forming a regular language which can be used by the agent to share knowledge with others. When viewed as a grammar, the rules form a proto-language whose structure mirrors that of the landscape.

To access the data stored in the grammar to reach a resource, the agent will need to generate a sequence of landmarks from the grammar which will form a path that the agent can follow. This is done by using the grammar to generate a list consisting of only landmark names. The starting rule for this expansion is a grammar rule in which the left-hand side is $goto(resource)$, where $resource$ is the type of resource the agent wishes to locate. The rule used is the one that leads to the resource which the agent most recently visited, which implies that rules the agent has acquired through exploration are used in preference to those gained linguistically. Symbols of the form $locX$ are used as non-terminals in the parsing and landmark names as terminals. Expansion takes place using an A* search with a metric based on the length of the path (as measured by the number of landmarks).

Agents utilise the grammar when they are called upon to share their knowledge linguistically with others, but instead of following the path it is passed to the other agent, who will store it in the form of the first rule given above. There are a few minor but important changes to how the agents use the grammar when generating a path for another agent which ensure that hearsay and misinformation are not passed onto other agents. These changes involve only allowing rules which have been formed through exploration to be used when communicating information to another agent and then only the subset of these rules that refer to resources that have been recently visited. These changes, which maximise the chances that a resource to which an agent is directed is actually there, were found to significantly improve the performance of communicating agents in our earlier research (Kazakov and Bartlett, 2004).

The deliberately impoverished representation of geographical knowledge that is used here might be found to be of use in other agent applications where navigation is needed. A particular strength of the representation is that it allows for a certain amount of generalisation to occur. As locations are not precisely specified as coordinates, but

rather they are defined in terms of the landmarks visible from that position, positions close to each other may be viewed as the same location by the algorithm, which allows a degree of abstraction about an agent's position. For example, in a domain involving robots navigating a building, it may not be important exactly where in a room a robot is for a navigational planner, rather which room the agent is in may be much more significant at this level. Landmarks too can provide for a form of generalisation. In the current work, landmarks have unique names, but there is no reason why this must be the case. If different landmarks had the same name, generalisation of any structural regularity of the environment would be possible, indeed inevitable, without an exhaustive search. Returning to the robot example, an identical series of visual or radio beacons leading to each door of a given room could be used, allowing a generalised concept of a path heading into that room to be formed. Other features, such as the minimal amount of information required to build up a series of routes, and the ability to associate scores rating the usefulness of particular rules, may also make this representation potential useful in other domains.

The representation of knowledge and its utilisation are discussed in more detail in our previous work (Bartlett and Kazakov, 2003).

5 Study of the Role of Environment

Intuitively, the benefits to a community of the ability to use language in this scenario might seem obvious. However it is not inevitable that the capacity to share resource locations should lead to an increase in a population's ability to survive. It is conceivable that spreading knowledge of resources could lead to much faster depletion of those resources and hence starvation, or that a population could be created in which all individuals had sufficient resources to survive but insufficient to reproduce.

The benefits of a particular altruistic policy to a community of agents can be measured through conducting experimentation in which a population of agents is placed within an environment and the simulation is allowed to run for a period equivalent to several generations. The simulation is performed three times with different populations of agents, one of which is able to communicate, one of which is able to share resources and a final population which is entirely selfish. Agents in populations in which language use is not allowed still use the information gathered about the environment for personal navigation to make for fairer comparisons between the populations. Any survival or reproductive benefit associated with the use of a strategy should be manifested in the population size, in this way we study language evolution as a form of multi-agent learning which improves a population's performance as measured by its ability to utilise resources.

In the present work, the environments in which agents

were placed had two of their structural properties changed in order to evaluate the effect that these had on populations implementing the various altruistic policies. A basic environment shape was defined, with four food resources equally spaced across the top of the environment and four water resources across the bottom as shown in figure 1. This map was then transformed along two dimensions to produce a series of maps. The distance between resources of the same kind was one factor that altered between experiments with the other factor being the distance between the different types of resource. Three values of each distance were chosen based on indications from our earlier work (Bartlett and Kazakov, 2004), 1, 3 and 5 squares for the distance between resources of the same kind and 3, 5 and 7 squares for the distance between resources of differing types. Each pair of distances were used in a map leading to a total of 9 environments. Figures 1 and 2 are two examples of the maps formed. Note that the number of resources in each environment is kept as a constant, as is their relative positioning, and that the different distances are thus achieved by altering the size of the environment. To ensure valid comparisons between experiments, the same number of agents were used in each environment.

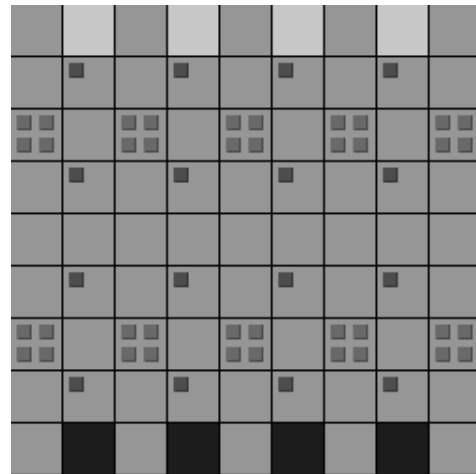


Figure 1: An environment with a space of one square between resources of the same type and seven squares between resources of different kinds. The coloured environment squares show where resources are located (food at the top, water at the bottom), and the smaller squares show the agents (in groups of four) and landmarks (uniformly spaced).

6 Results and Discussion

The results, as shown in figure 3, demonstrate that the population in which language is used performs significantly better than the other populations in most cases studied. As would intuitively be expected, the greater the

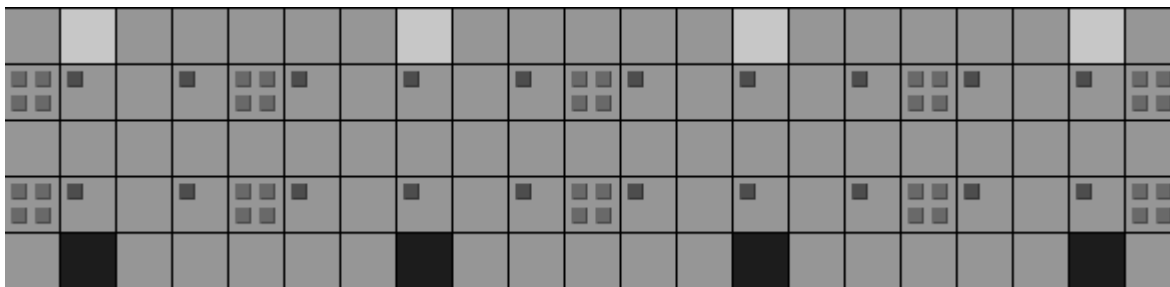


Figure 2: An environment with spacing of 5 squares between like resources and three between differing ones. The same rendering as figure 1 is used.

distance between food and water the fewer agents manage to survive regardless of strategy. However, though the resulting size of the language-using population decreases as this distance increases, the relative benefits of language increase, as can be seen by the ratio of the size of the language-using population to the selfish population (which can be viewed as a baseline for comparison).

The size of the final population also appears to be dependant on the spacing between resources of the same kind. Again the general rule appears to be that the greater the distance, the lower the population which manages to survive though the effect is not as pronounced as changing the distance between food and water. Language use also appears to increase in effectiveness with increasing distance for this variable.

The reason for the decrease in the size of population as the food-water distance increases is quite obvious, and is simply due to the greater difficulty in an agent managing to find both resources before dying of hunger or thirst. This reveals why language use is more beneficial in situations where the distance is greater. By using language, the complexity of the search for resources is simplified, as agents are able to gain information on resource location without having to go through the process of exploration. In effect, the use of language manages to parallelise the searching for resources. The effect of language is likely to be particularly pronounced in later generations when agents born into non-communicating populations have to locate resources through exploration as their parents did, but communicating agents will most likely be able to gain this knowledge linguistically almost immediately from their parents.

It is less immediately obvious that the distance between resources of the same type should influence the population dynamics though there are actually two ways in which the spacing between resources is likely to do so. Firstly, as stated earlier in this paper, resources are exhaustible with use. This means that occasionally agents will need to migrate from one source of food to another (similarly with water). At this point the distance between food sources becomes relevant. When the resources are separated by one square, it is almost inevitable that a move from an exhausted food source will lead to a po-

sition in which a new resource can be seen, whereas the task of locating a new resource becomes a matter of quite extensive exploration when the next closest resource is 5 squares away. Secondly, the difficulty of locating a resource initially in the environment increases in complexity as the distance between resources of the same kind increases. To see why this is so, consider two environments with the same distance between food and water but different distances between food and food (and therefore water and water). The number of squares from which food is visible in both environments is constant, being one square in each direction away from the food. However the number of squares in the environments are different and hence the environment in which food is more spaced out will need a greater proportion of the squares to be searched in order for resources to be found. Again, the reason that language performs better is due to the fact that it manages to reduce the amount of searching that must be done to locate resources.

There are two other effects that may also explain some of the benefits that language use brings. Firstly, populations using language have a tendency to cluster into a smaller part of the environment: the majority of agents tend to become reliant on a subset of the resources available. This occurs as the process of agents informing others of the resources of which they know results in most agents becoming aware of the same resources and routes between them. By clustering themselves into a reduced part of the environment, the probability of agents who are able to mate meeting is increased and hence the expected birthrate will also be increased. Secondly, the process of being able to learn routes through language produces an effect that results in shorter, more efficient routes becoming used. As communicating agents are able to gain information through both exploration and language, they are able to learn of several different routes to the same location. The details of the navigational planning algorithm ensure that agents will tend to take the shortest route which they are able to, and hence agents in communicating populations will tend to travel shorter routes than those agents in other populations, which reduces their chances of dying while travelling between resources and increases the time for which they are able to mate.

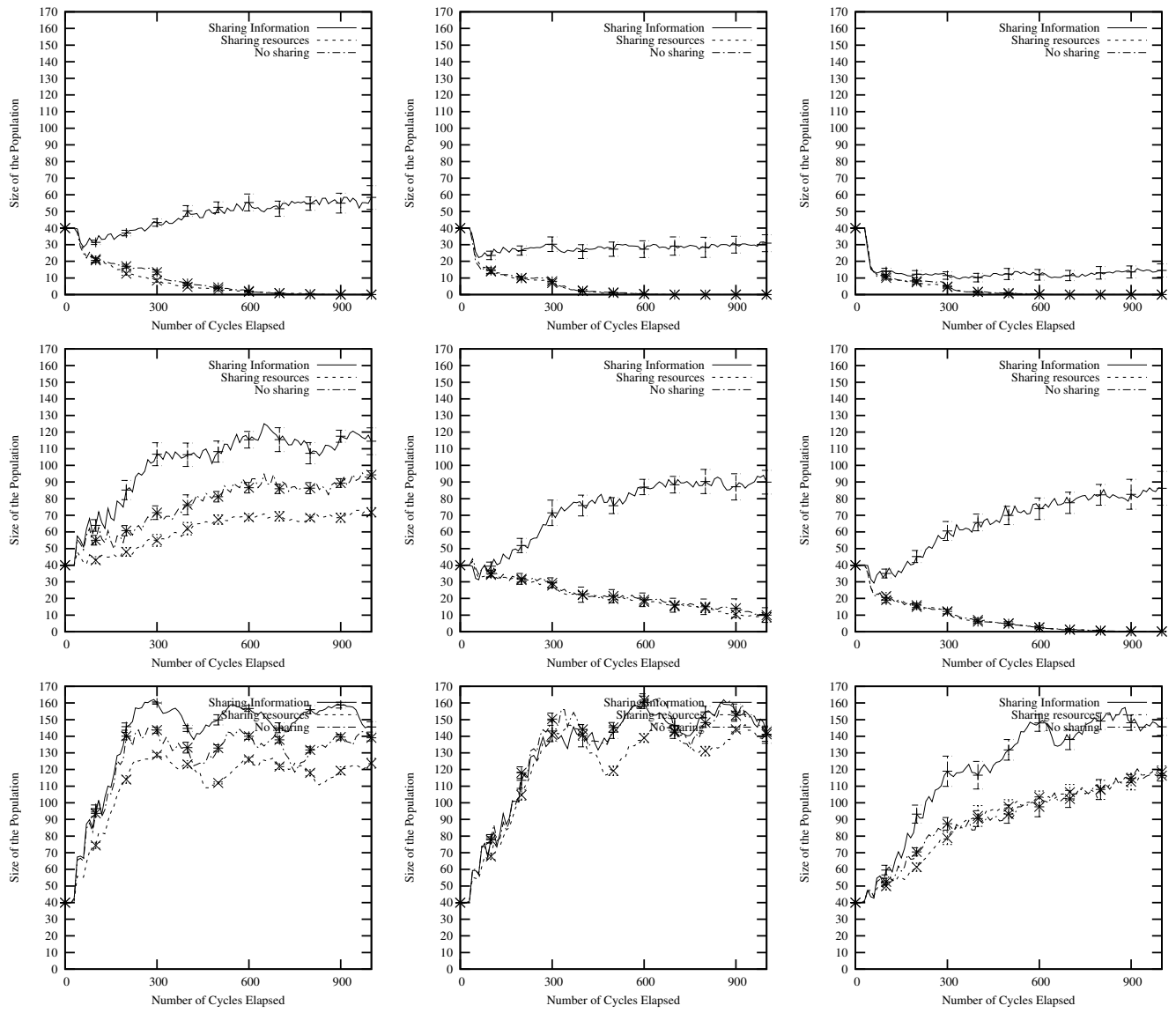


Figure 3: Comparison of sharing information, sharing resources and selfish behaviour. The distance between resources of different kinds increases from bottom to top, from 3 to 5 to 7 squares. The distance between resources of the same kind increases from left to right, from 1 to 3 to 5 squares. All results are averaged over 10 runs.

In all the experiments conducted, populations which shared resources performed no better than the baseline behaviour of not sharing at all, and in some cases did significantly worse. This serves to illustrate that the improved performance of the language-using community is not just down to the altruism of the communicating agents. The poor performance of the population who share resources can probably be best explained through the fact that sharing resources can lead to a population in which resources tend to become more evenly spread throughout the population. This creates a population in which fewer agents are able to reproduce, and those agents that are created through mating begin life with lower resource levels and thus a worse chance of surviving. One conclusion that might be drawn from the evidence produced is that agent populations would perform better if they concentrated on gathering more resources and were less concerned about how those resources were distributed between them.

A further point that should be mentioned with respect to the performance of agents who share resources is that they take noticeably longer to recover from the depletion of a resource than do agents who are not sharing resources. This can be seen in the graphs towards the bottom left-hand corner of figure 3. The periodic increase and decrease in population size seen in these graphs is characteristic of resources becoming depleted (Kazakov and Bartlett, 2004). It can be seen that in these experiments, the population size of the agents who share resources falls for longer than the other populations. This is caused by exploration to locate new resources being delayed by agents, who are content to draw energy from others rather than find a new permanent source of resources. When agents do eventually begin to explore, their collective resource levels are much reduced by this period of inactivity and it takes agents longer to regain a state conducive to procreation.

In passing, it can perhaps be noted that the fact that the periodic increases and decreases in population size are not seen in some graphs is due to the fact that in the experiments in which this is the case the population size is much smaller in relation to the environment size. This means resources are not being used to their full potential and thus the time at which they become depleted varies from resource to resource and from trial to trial, blurring the cyclic behaviour to the point where it cannot be seen in the graphs.

7 Conclusion and Future Work

This paper has outlined a multi-agent system built specifically to study the phenomenon of language evolution and has presented a range of experiments that have been carried out in order to study the influence of certain features of the environment upon the usefulness of language to a community of speakers. Specifically, we have carried out investigations in which two different types of the distance

between resources was varied. Our results show that in all the cases considered, language use was never detrimental to a population and became increasingly advantageous compared with other policies as distances between resources increased. Sharing resources, on the contrary never offered any benefit over selfish behaviour and actually proved to be a disadvantage when resources were most easily discovered.

It is our future intent to assess the effect that other features of the environment may have upon the usefulness of language to a population, for example varying the spacing of landmarks or introducing impassible terrain. Through experimentation we hope to assess the way in which features of the landscape such as these affect the size of the population and the segmentation of the linguistic community, and draw parallels between this and speciation and cultural division in the natural world.

When completed, this research should shed light on when exactly language evolution of this type should be expected to provide some benefit and hence its use be selected for by natural selection. Our intuition is that we will find the conditions in which language use is encouraged to be similar to those found at the time when primitive language use is believed to have begun in humans. Indeed the fact that in the present research, language use was found to be more beneficial as distance between resources increased, may be considered alongside the fact that language may well have first arose in early humans at the same time as they moved from a jungle habitat to the savannahs of east Africa. Such a change would have taken them from an environment in which food and water were quite abundant into one in which resources could frequently require many miles of travel to reach. Our research suggests that this is exactly the kind of environment in which language users would be likely to have a reproductive advantage.

References

- M. Bartlett and D. Kazakov. Social learning through evolution of language. In P. Liardet, P. Collet, C. Fonlupt, E. Lutton, and M. Schoenauer, editors, *Proceedings of 6th International Conference on Artificial Evolution, EA'03*, pages 340–351, Marseille, France, 2003.
- M. Bartlett and D. Kazakov. Replacing resource sharing with information exchange in co-operative agents. 2004. Submitted to the Third Conference on Autonomous Agents and Multi-Agent Systems, New York.
- E. J. Briscoe, editor. *Linguistic Evolution through Language Acquisition: Formal and Computational Models*. Cambridge University Press, 2002.
- R. Dawkins. *The Extended Phenotype*. Oxford University Press, 1982.

- W. D. Hamilton. The genetic evolution of social behaviour I. *Journal of Theoretical Biology*, (7):1–16, 1964.
- M. D. Hauser, N. Chomsky, and W. T. Fitch. The faculty of language: What is it, who has it, and how did it evolve? *Science*, 298:1569–1579, November 2002.
- D. Kazakov and M. Bartlett. A multi-agent simulation of the evolution of language. In M. Grobelnik, M. Bohanec, D. Mladenec, and M. Gams, editors, *Proceedings of Information Society Conference IS'2002*, pages 39–41, Ljubljana, Slovenia, 2002. Morgan Kaufmann.
- D. Kazakov and M. Bartlett. Benefits of sharing navigational information in a dynamic alife environment. 2004. Submitted to the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9), Boston, Massachusetts.
- D. Kazakov and D. Kudenko. *Machine Learning and Inductive Logic Programming for Multi-agent Systems*, pages 246–270. Springer, 2001.
- S. Kirby. chapter Learning, Bottlenecks and the Evolution of Recursive Syntax. Cambridge University Press, 2002.
- J. O’Keefe and L. Nadel. *The Hippocampus as a Cognitive Map*. Oxford University Press, 1978.
- M. Oliphant. *Formal Approaches to Innate and Learned Communication: Laying the Foundation for Language*. PhD thesis, Department of Cognitive Science, University of California, San Diego, 1997.
- L. Steels. The spontaneous self-organization of an adaptive language. In K. Furukawa, D. Michie, and S. Muggleton, editors, *Machine Intelligence 15*, pages 205–224, St. Catherine’s College, Oxford, 1999. Oxford University Press. Machine Intelligence Workshop: July 1995.
- H. Turner and D. Kazakov. Stochastic simulation of inherited kinship-driven altruism. *Journal of Artificial Intelligence and Simulation of Behaviour*, 1(2):183–196, 2002.
- W. H. Zuidema and P. Hogeweg. Selective advantages of syntactic language - a model study. In *Proceedings of the Twenty-second Annual Conference of the Cognitive Science Society*, pages 577–582, Hillsdale, USA, 2000. Lawrence Erlbaum Associates.

Baselines for Joint-Action Reinforcement Learning of Coordination in Cooperative Multi-Agent Systems

Martin Carpenter

*Department of Computation

UMIST

Manchester

M.Carpenter@umist.ac.uk

Daniel Kudenko

†Department of Computer Science

University of York

United Kingdom

kudenko@cs.york.ac.uk

Abstract

A common assumption for the study of reinforcement learning of coordination is that agents can observe each other's actions (so-called *joint-action* learning). We present in this paper a number of simple joint-action learning algorithms and show that they perform very well when compared against more complex approaches such as OAL (Wang and Sandholm, 2002), while still maintaining convergence guarantees. Based on the empirical results, we argue that these simple algorithms should be used as baselines for any future research on joint-action learning of coordination.

1 Introduction

Coordination is a central issue in multi-agent system (MAS) development, and reinforcement learning of coordination has been intensively studied in recent years. In this paper we focus on coordination in cooperative MAS, i.e., where all agents share the same objective.

Coordination learning algorithms that have been developed for this task can be classified in two categories. The first category comprises those learning algorithms that make no assumption on whether agents can communicate, observe each other's action, or even be aware of each other's existence (so-called *independent* learning techniques, see e.g. (Kapetanakis and Kudenko, 2002)). In the second category there are those algorithms that assume that agents can observe each other's actions and use these observations to improve coordination (so-called *joint-action* learning techniques (Claus and Boutilier, 1998)). The latter case being considerably easier to handle, most research to date has been focused on joint-action learners.

However, we argue that there exist very simple coordination learning techniques for agents that can observe each other's actions, and that these algorithms perform comparably well while being guaranteed to converge towards the optimum. We strongly suggest that the techniques presented in this paper should serve as baselines in any further research on joint-action learning of coordination in cooperative MAS.

The paper is structured as follows. We first describe two formalisms that are commonly used to represent coordination problems: single-stage games and multi-stage Markov games. We then present simple algorithms for the learning of coordination in such games and evaluate them empirically on a number of difficult coordination games from the literature. We also compare the performance of

these simple learning techniques to the OAL algorithm (Wang and Sandholm, 2002), that has been widely cited in recent research. In addition to the simple learning algorithms, we also present an improvement based on statistical confidence measurements, and discuss its applicability. We conclude the paper with a summary and an outlook to the future of joint-action learning of coordination.

2 Cooperative Games

Games are a common and highly suitable way to represent cooperative coordination problems. Depending on the problem to be modelled, there are two main representation options: single-stage games and multi-stage (or *Markov*) games. In this section we briefly introduce both.

2.1 Single Stage Games

Single stage games are relatively simple models of interaction, and have been mainly used in the context of game theory. A number of players simultaneously perform an action each from a set of possible actions, after which the game is over and each player receives a reward. In the cooperative case all players receive the same reward. More formally:

Definition 1 *An n -player single-stage co-operative game Γ is a tuple (A^1, \dots, A^n, r) for $k \in \{1 \dots n\}$ where A^i is the discrete action space of player i , and $r : A^1 \times \dots \times A^n \rightarrow \mathbb{R}$ is the payoff function mapping actions to numerical reward values. Each player shares identical payoff functions.*

Single stage games with only two players can be represented as a rectangular matrix of numbers. An example of

a game represented in such fashion is given in Figure 1: each of the two players can choose between two actions a and b . The matrix entries determine the payoff. For example if Player 1 plays b and Player 2 plays a , then they both receive a payoff of 5.

		Player 1	
		a	b
Player 2	a	3	5
	b	0	10

Figure 1: A simple game matrix.

A number of single-stage games have been defined in the research literature that pose specific challenges to learning of coordination, and we have used these to evaluate our approaches. These games are described below.

2.1.1 The Climbing Game

A variant of this single-stage game (shown in Figure 2) has been introduced in (Claus and Boutilier, 1998). It poses a difficulty because the optimal joint action $((a, a)$ with a payoff of 80) has an associated high miscoordination penalty, and therefore a regular independent Q learning approach does not converge to the optimum (Kapetanakis and Kudenko, 2002).

		Player 1		
		a	b	c
Player 2	a	80	-60	0
	b	-60	40	30
	c	0	0	20

Figure 2: Climbing Game

2.1.2 The Penalty Game

This game (shown in Figure 3), also introduced in (Claus and Boutilier, 1998), has two optimal joint actions. The major difficulty is that a mis-coordination in the choice of the optimum leads to a penalty.

		Player 1		
		a	b	c
Player 2	a	40	0	-40
	b	0	20	0
	c	-40	0	40

Figure 3: Penalty Game

2.1.3 The Stochastic Penalty Game

This game is equivalent to the penalty game above. However, the payoffs are stochastic. Each joint action may

lead to one of two payoffs with probability 0.5 each.

		Player 1		
		a	b	c
Player 2	a	80/120	-30/30	-80/-120
	b	-30/30	0/40	-30/30
	c	-80/120	-30/30	80/120

Figure 4: Stochastic Penalty Game

2.2 Markov Games

In contrast to single-stage games, multi-stage (or Markov) games do not have to end after just one round. Players simultaneously perform actions, and after each round, a new state is reached. The new state is determined probabilistically based on the actions of the players. In each state the players receive a payoff.

Definition 2 An n -player Markov game is a tuple (S, G, p, r) where S is the discrete state space of the game, G is a set of single stage games, one for each state in S , and $p: S \times A^1 \times A^n \rightarrow \Delta$ is the transition probability map, where Δ is the set of probability distributions over the state space S . $r: S \rightarrow \mathbb{R}$ is the payoff function that maps states to numerical reward functions.

A Markov game can be considered as a collection of single state games linked by a function governing transitions between the games. They can be drawn as a set of matrices linked by arrows with the transition probabilities marked on them.

As in the case of single-stage games, a number of multi-stage can be defined that pose specific coordination challenges. Two representative games that we used for evaluation are presented below.

2.2.1 C&B Game

This game was introduced in (Chalkiadakis and Boutilier, 2003) and is shown in Figure 5. At each state each player has a choice of two actions. The transitions on the diagram are marked by a pair of corresponding actions, denoting player 1's move and player 2's move respectively. "*" is a wild card, representing any action. States that yield a reward are marked with the respective value. All other states yield a reward of 0.

2.2.2 Misleading Game

The difficulty of the game shown in Figure 6 is that actions that lead to early high payoffs are not part of the best policy, and are thus misleading.

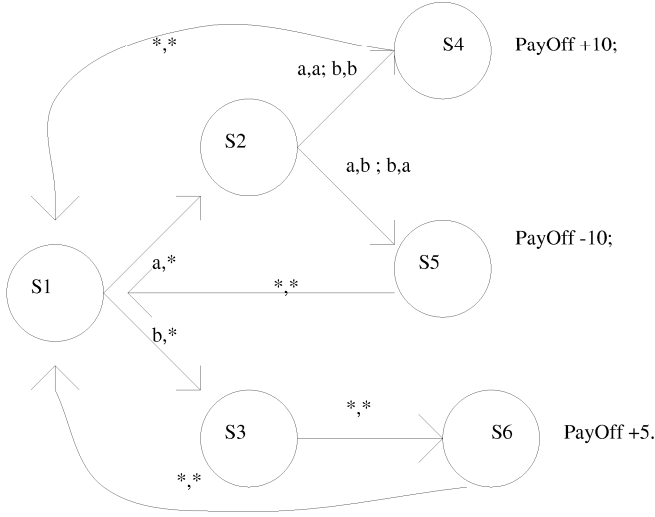


Figure 5: C&B Game

3 Simple Learning of Coordination

There are many approaches to learning coordination in cooperative games (for example, (Littman, 1994; Claus and Boutilier, 1998; Nowe et al., 2001; Wang and Sandholm, 2002)). In this section we present simple joint-action learning techniques that will then be evaluated.

3.1 Simple Coordination Learning for Single Stage Games

3.1.1 Basic Approach

Our proposed learning algorithm for single stage games has a very simple form:

1. Choose between the possible actions at random
2. Observe the joint action played and reward recieved
3. Update the average reward obtained for the joint action just played
4. Repeat until stopped, or some preset number of repeated plays has completed.

The algorithm then outputs the agent's action that is part of the joint action with the highest average payoff. If two joint actions have the same average payoff then it chooses the action which was played first.

For a game with constant payoffs this algorithm is guaranteed to find the optimal joint action if given enough time. Formally, if the game matrix has size $n \times m$ with o optimal actions, the algorithm has a $1 - (\frac{nm-o}{nm})^k$ chance of converging to a optimal joint action in k moves. Note that this convergence speed is independent of any structural characteristic of the game besides the number of possible joint actions. For example in the climbing game presented below it will take an average of 25 moves to achieve 95% convergence and in the penalty game (which

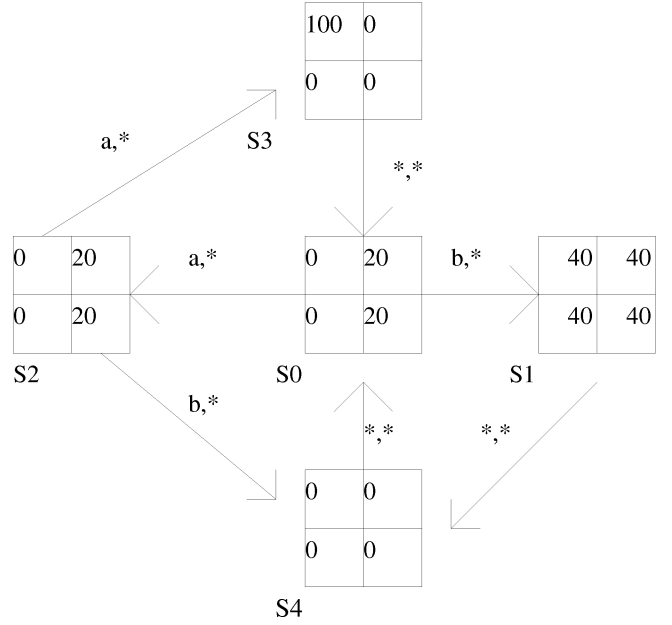


Figure 6: Misleading Game

has two optima) it will take an average of 12 moves to achieve this. For games with stochastic payoff functions the time taken to find an optimal answer will scale linearly with the difficulty of finding the true averages of the payoff functions. For example in the stochastic penalty game either result from the optimal joint actions is optimal and so it solves it as fast as it does the penalty game. The simple stochastic game is harder to give exact figures for. In 45 moves it will on average have visited the optimal result 5 times with a bit over 50% chance of having found the right answer.

3.1.2 Performance Enhancement

While the basic learning algorithm already shows very good performance, there is a simple way to further improve it's performance by modifying the action selection technique employed during learning. Whilst purely random exploration is guaranteed to visit every joint action arbitrarily often as the number of moves tends to infinity, it is rather inefficient.

In order to improve performance we use the following action selection algorithm instead of purely random action choice: At each step choose the action for which the smallest number of joint actions has been observed at least once. If there is more than one action with this property then the choice is random.

The idea behind this algorithm can be visualized on a 2×2 game matrix: After the first move has been played, both agents will switch action forcing them to play in the diagonally opposite corner. Then once they randomly choose one the previously unplayed joint actions they are forced to choose the remaining unplayed joint action.

On average this will take 5 moves to visit every square

in a 2 x 2 grid, and thus solve a single state 2 x 2 game with deterministic payoffs. This is significantly faster than using purely random move selection.

There is however the potential for this algorithm to deadlock on larger grids, as shown in the following example. Visited joint actions are marked as X's, previously unvisited joint actions as O's.

After 3 moves without loss of generality the agents reach the following position:

$$\begin{bmatrix} X & O & O \\ O & X & O \\ O & O & X \end{bmatrix}$$

The agents then choose a random move and might reach the following state:

$$\begin{bmatrix} X & X & O \\ X & X & O \\ O & O & X \end{bmatrix}$$

From that state they could end up in:

$$\begin{bmatrix} X & X & O \\ X & X & O \\ O & O & Z \end{bmatrix}$$

when the agents are stuck playing joint action Z forever.

No doubt more complicated deadlocks are possible on larger grids. Hence we need a mechanism to avoid these situations: if the algorithm generates the same action twice in a row without having invoked a random move, then it has to play a random action instead.

Whilst these problems mean that performance isn't as good as in the 2 x 2 case, the enhanced action selection is still an improvement over the basic technique. Since it also involves little extra complexity it was used in the experiments below.

The advantages of the modified action selection algorithm over random action choice are not as big in games with stochastic pay offs. In such games each joint action has to be visited multiple times, rather than just once, in order to solve the game. Random action choice leads to a more uniform distribution of actions.

3.2 Simple Coordination Learning for Multi Stage Games

For the multi-stage case, our proposed algorithm is based on Q learning C.Watkins (1989), and uses random action selection during the learning phase. More formally, on observing the current joint action A , its payoff, and the current state s , an agent first updates the Q value for A and s :

$$Q(s, A) = R(s, A) + \gamma \sum_{s' \in S} T(s, A, s') \max_{a' \in A} Q(s', a')$$

Here n is the number of times the joint action A has been seen, $R(s, a)$ its average payoff for action a in state s so far, γ is the discount factor (we chose 0.9), and T holds the transition probabilities so far observed. When learning is finished (after a fixed number of iterations),

each agent returns the action that is part of the joint action with the highest Q value.

Because the Q values are calculated over the joint actions, with each agent calculating identical Q value tables, this is exactly analogous to the case of single agent Q learning and convergence to an optimal action choice is therefore guaranteed C.Watkins (1989).

While it is difficult to generally compute the time required for the agents to converge to the optimal joint action, we present this computation for the C&B game from Figure 5.

Each choice of moves in a three state cycle can be viewed as single choice between $2^3 = 8$ different possible moves. The choice of move in the final state is however for this game irrelevant as you always return to the start state, so there are only four different actions to consider: aa, ab, ba, and bb. Since action choice at each state is independent of the choices made at previous states for this algorithm then this equivalence is clear. The factor of three is simply because you're treating three separate moves as single ones. This could take quite a long time except that such games typically have numerous identical pay offs if considered in this way. For instance the game below in figure 5 is equivalent to the following 4 by 4 game (with the actions being considered as aa, ab, ba, and bb).

		Player 1			
		aa	ab	ba	bb
Player 2	aa	10	-10	-10	10
	ab	10	-10	-10	10
	ba	5	5	5	5
	bb	5	5	5	5

Thus the multi-state learning algorithm will have a $(1 - (\frac{12}{16})^{(\frac{4}{3})})$ chance of converging in k moves. For example, it will reach 94.4% convergence in 30 moves.

4 Empirical Evaluation

In this section we show that the algorithms proposed in the previous section perform very well on a wide range of problems from the research literature. For further evidence, we compare the performance of the algorithms to the OAL algorithm (Wang and Sandholm, 2002), a widely cited learning technique that is applicable to both single-stage and multi-stage games. We do not present details of the algorithm here, but rather refer the reader to the original paper. Note however, that OAL is much more complex than the simple approaches presented in Section 3.

For our experiments OAL's results were averaged over 1000 runs, the simple algorithms, which run a lot faster, over 10,000 runs.

In the result graphs below, the single-stage game learning technique is denoted by Baseline+ and the multi-stage game algorithm by MSBaseline.

4.1 Evaluation on Single-Stage Games

We evaluated the single stage game learning algorithm on the Climbing Game, the Penalty Game, and the Stochastic Climbing Game (described in Section 2.1. Figures 7, 8, and 9 show the probabilities of convergence to the optimal joint action as a function of the number of learning steps. The results show that the simple learning technique clearly outperforms OAL.

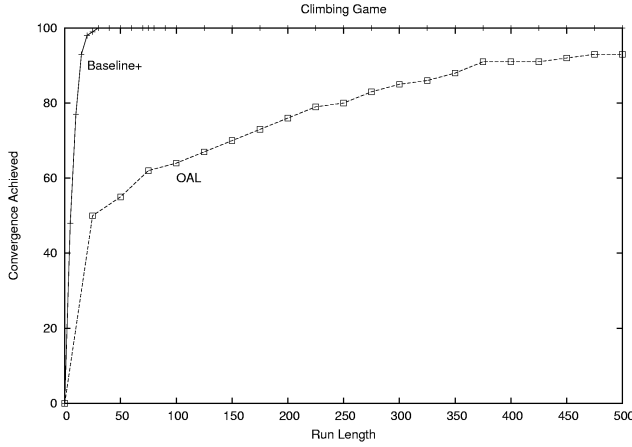


Figure 7: Evaluation on the Climbing Game

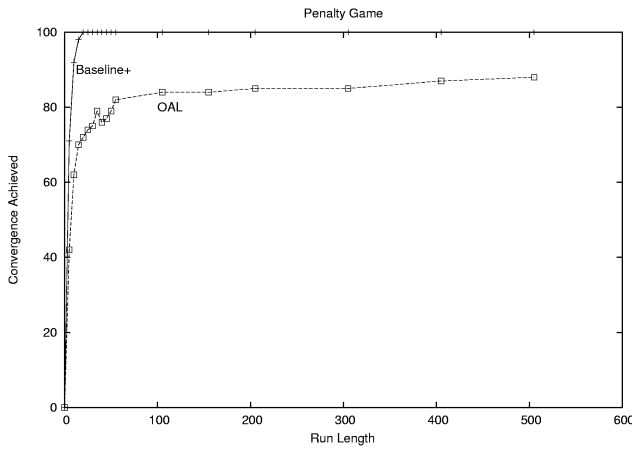


Figure 8: Evaluation on the Penalty Game

4.2 Evaluation on Multi-Stage Games

The multi-stage game algorithm has been evaluated on the C&B game and the Misleading Game from Section 2.2. The simple learning technique performs very well on both games, as shown in Figures 10 and 11.

5 Another Improvement

For reasons explained later, the `baseline+` algorithm currently only works for single state games (or multi state

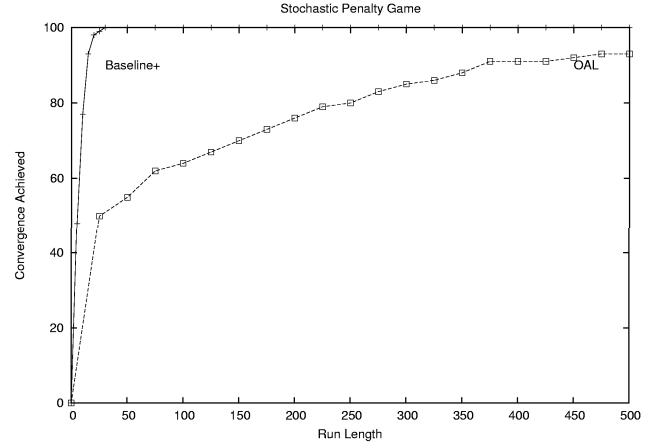


Figure 9: Evaluation on the Stochastic Penalty Game

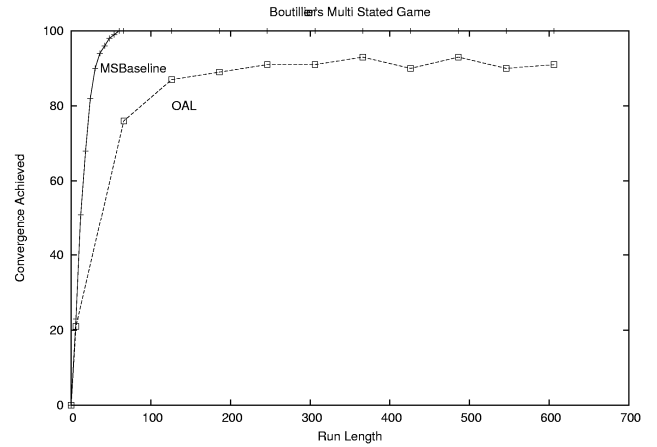


Figure 10: Evaluation on the C&B Game

games with non stochastic state transition functions) and so these are the type of game referred to in it's description.

While the algorithms presented above are extremely fast joint action learners for games with constant pay offs there is still a lot of room for improvement in games with stochastic payoffs. This can be seen by considering a 3×3 game with one payoff which gives 100, -20 and -20 with equal probability and where the other payoffs give 0. An ideal learner would concentrate on observing the 100/-20/-20 payoff until it was fairly sure whether it was bigger or smaller than the 0 payoffs and spend little time on the constant payoffs.

The first possibility is biasing action selection in favour of those joint actions whose payoffs were displaying the highest standard deviations and of which we were thus least sure about the true value. However this would perform badly in games where there was a payoff with very low expectation but high standard deviation. Clearly some measure of how good an observed payoff is should also be taken into consideration.

Parametric statistics are a useful tool to achieve this. The observed expected payoffs for each joint action will

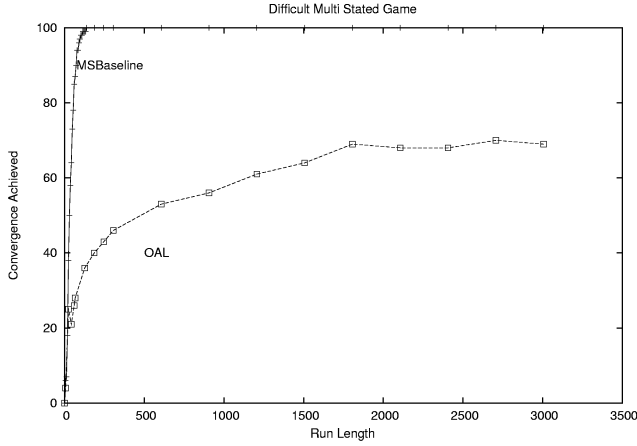


Figure 11: Evaluation on the Misleading Game

be distributed approximately normally, in a distribution which tends towards the normal distribution as the number of observations tends toward infinity, and the standard deviations of these distributions are easy to calculate. This involves several approximations but they can be generally be expected to be within reasonable range to the true value.

Since agents know both the means and standard deviations for these normal distributions, they can compare their means and say how likely it is that one payoff might have a true mean bigger than another's. This idea (hypothesis testing) is very common in statistics and is central to our algorithm.

The second inspiration for our algorithm was a desire to have a self terminating algorithm. Our aim was to allow the user to set a confidence limit, after which the algorithm will take as long as it needs to get that accurate an answer given the game it is learning. This idea contrasts with many current algorithms where the designer explicitly specifies in advance when the algorithm should stop learning, whether directly as in the case of the algorithms above, or indirectly via a decaying temperature setting as in many Q learning implementations.

If the structure of a game is unknown, it is hard to predict how long it will take a given algorithm to reliably solve it.

5.1 The Statted Learning Algorithm

Ideally, a user would choose a percentage indicating a degree of confidence that the joint action that the agents choose is the true optimum. Unfortunately this is not completely possible due to the approximations involved.

Our proposed algorithm works in two distinct stages and requires initialisation of the following variables:

- **SDTolerance** : how many standard deviations the user wants to test against to see whether an action could possibly be the optimal action or not. This de-

fines the desired confidence in the optimality of the resulting joint action.

- **ExplorationCutoff**: An integer constant that is explained below. This setting also effects the accuracy of the results obtained.
- **ToleranceLevel** : Explained below. This defines how close agents would let the expected values for a pair of joint actions be before they ceased to care about which joint action is selected from them.
- σ_0 : A number used to try and increase the accuracy of the estimates of the sample standard deviations. This is useful as otherwise one can get quite misleading sample standard deviations when trying to approximate distributions far from the normal.

Having received these inputs, the algorithm then performs an exploration phase followed by a phase of continuous exploitation which is ended when it outputs a joint action which it considers to be likely to be optimal.

5.1.1 Exploration

This stage is designed to achieve two things: that all joint actions have been observed and that the observed payoffs from the joint actions are in some way representative of the actual expected pay off from playing those joint actions.

This phase consists of playing random moves whilst tracking both the mean and the sample variance of the joint actions observed, which is given by $\hat{\sigma}_\mu^2 = \frac{S + \sigma_0^2}{n^2 - m^2}$. Here S is the sum of squared received rewards, σ_0 is defined above and n is the number of times this joint action has been played.

Basically σ_0 is used as a safety net: if a joint action has turned up the same value multiple times, then we still want it to have some standard deviation in case it has a second value that it can also take. This is only an issue with stochastic payoffs of the form $p/-p$, where each of the payoffs has probability 0.5. The means observed from more continuously distributed payoff functions will approximate much faster to normal distributions.

When every joint action which has been observed at least once, has been seen at least **ExplorationCutoff** times, exploration ends and the algorithm moves into exploitation.

5.1.2 Exploitation

First, the joint action with the highest currently observed reward is identified.

Then form a set of all possibly optimal joint actions OA is formed by including joint action j_i if $\frac{m_1 - m_j}{\sqrt{\sigma_1^2 + \sigma_j^2}} \leq \text{SDTolerance}$. This is a simple one tail hypothesis test on the sample means of the payoff distributions.

If OA has only one member then the action that is part of that joint action is returned as the proposed optimum.

The algorithm above has the problem that it does not terminate when there are two joint actions with identical long term payoffs. Also, it could take a very long time if there were two joint actions with very similar pay offs. In order to overcome these problems we introduced a test for whether all the means for the possibly optimal actions were within a given tolerance limit of each other.

This test is passed if for every $m_j \in \text{OA}$:

$$(m_1 - m_j) + \text{SDTolerance} \times \sqrt{\sigma_1^2 + \sigma_j^2} \leq \text{TL}$$

Here m_1 is again the action currently believed to be optimal and TL is the tolerance level as provided as input into the algorithm. If this test is passed then the differences between the expected payoffs of the joint actions in OA are considered to be insignificant and you can output a member of OA as the optimal action. Various techniques for this choice are possible, going for the one with the lowest standard deviation seems a good idea.

If both tests of these tests fail, then the algorithm continues by playing the joint action in OA which has the highest standard deviation. The results for that joint action are then updated in line with the observed result, and the exploitation phase continues until one of these tests is passed and an answer is produced.

		Player 1		
		a	b	c
Player 2	a	160/-40/-40	0	0
	b	0	0	0
	c	0	0	0

Figure 12: Simple Stochastic Game

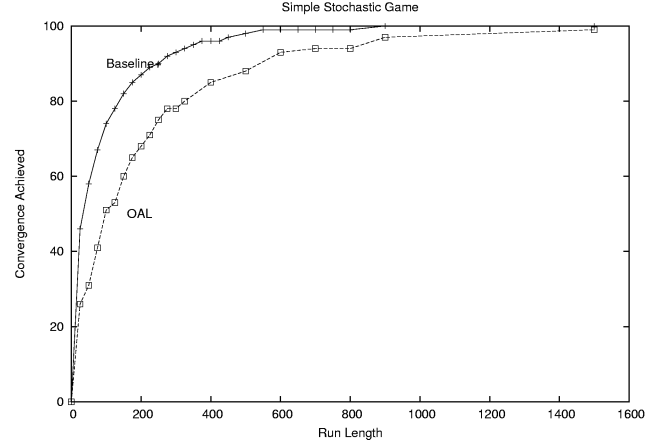


Figure 13: 'Simple' Stochastic Game

5.2 Statted Baseline Evaluation

The statted algorithm by it's very nature doesn't produce a graph of convergence achieved against time, but rather individual points which correspond to a fixed choice of settings. Each point has an average convergence and a average speed taken to achieve this level of convergence. Hence tables rather than graphs are used to present the results.

The statted Baseline is tested separately as it's use in game which don't feature large stochastic elements in their pay off functions is pointless.

5.2.1 The 'Simple' Stochastic Game

$$\begin{bmatrix} 160/-40/-40 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

This game (shown in Figure 12) is designed to test the algorithm's ability to cope with having to pick out an optimal action from among some discouraging payoffs. The issue is how long it takes for the agents to play the 160/-40/-40 enough times in order to realise that it is the optimal action. The statted learning algorithm was inspired by this game and one would expect it to perform well on it.

The evaluation results are shown in Figure 13 and in the tables below. These are given in some detail in order to contrast the effects of various settings upon its performance.

If you put: SD = 1, Min SD = 1, Sens = 0.1 we get:

Exploration Cut Off	Time Taken	Convergence
3	54	57
4	69	78
5	80	84
6	90	87
7	101	89
8	111	90
9	122	92
10	133	92
11	144	93
12	155	96

Now Fixing Exp Cut off = 10, And varying SDtol:

SDtol	Time	Convergence
0	126	86
0.2	126	84
0.4	127	86
0.6	128	88
0.8	130	91
1	133	93
1.2	137	94
1.4	142	96
1.6	148	96
1.8	154	97
2	163	98
2.2	171	98
2.4	181	99

If Exp Cut off is 5 then:

SDTol	Time	Convergence
0	67	75
0.5	69	78
1	80	85
1.5	94	89
2	116	90
2.5	142	91
3	178	91
3.5	220	91
4	267	91
4.5	318	93
5	379	93

The fastest settings tested were:

ToleranceLevel = 1, MinSD = 0; SDTol = 2, ExpCutOff = 7 giving 95% Convergence in an average of 126 moves.

If everything is kept the same except for setting ECOff = 10, the algorithm yields 98% convergence in 159 moves. Setting ECOff = 15 results 99% convergence in 212 moves.

Clearly, there is a risk that the joint action (a,a) comes out of the exploration phase with a large negative expectation and then being ignored subsequently.

5.2.2 The Stochastic Penalty Game

This game was defined in Section 2.1. It is not that interesting as an evaluation of the stated algorithm since either payoff from the optimal joint actions is higher than any other achievable pay off. Thus once the agents have played the optimal joint action once they know that it is optimal.

This game does however illustrate one big performance issue with the stated learning algorithm. If the tolerance level is set too low it can take a very long time to decide that two joint actions give equivalent pay offs.

For example, if ExpCutoff = 6, SDTolerance = 2, MinSD = 5, and ToleranceLevel = 1, 100% convergence is reached in 6300 steps. If the tolerance level is increased to 10 the required number of steps to guarantee convergence drops to 191. When set to 15 this further drops to 129 steps.

And with the settings of ExpCutoff = 20, increasing SDTolerance to 4 gives a success rate of 99% in 85 moves.

This is typical of games where there are several optimal joint actions with equal payoff. A lot of the average time taken in the simple stochastic game was due to the agents having decided that the optimal pay off was bad and then having to decide that the eight payoffs yielding zero were equal.

A large factor in causing this problem is the decision to have a minimum standard deviation in the algorithm. If this wasn't used then constant payoffs would have 0 standard deviation and the algorithm wouldn't take any time at all to solve these problems. In some cases the MinSD factor should therefore be removed from the algorithm entirely.

5.2.3 Two Difficult Tests

The game in Figure 14 has been designed as a tough test of an algorithm's ability to tell quite similar stochastic payoffs apart. In this setup the Baseline+ algorithm achieves about 95% convergence after 5500 runs; whereas the stated algorithm only takes about 2700 moves to reach that level. One would expect this, as the stated baseline concentrates its action choice on distinguishing the 3 close payoffs (6/-4 and the 5/-4's).

		Player 1		
		a	b	c
Player 2	a	6/-4	3/-3	-6/4
	b	3/-3	5/-4	3/-3
	c	-6/4	3/-3	5/-4

Figure 14: Difficult Stochastic Game

Replacing one or two of the 5/-4 payoffs with 6/-4 payoffs should favour the baseline over the stated baseline as the stated baseline will worry about checking to see if they are the same, whereas the baseline will just have two or more targets instead of only one.

And indeed if one of the 5/-4's is replaced with a 6/-4 then the Baseline+ algorithm reaches 95% convergence in about 2000 moves, and the stated algorithm performance is very similar.

If the final 5/-4 is replaced with another 6/-4 then, one would expect the baseline to dominate and it does: it reaches 95% convergence in only 400 moves. The reason for this is that there aren't any payoffs remaining close to the 6/-4's to distract it. The stated algorithm becomes very dependent on its sensitivity setting: if you set it to 0.5, then it achieves 94% in 2872; if you set it to 1 then it achieves 93.5% in 1032 moves. Clearly it is spending most of its time deciding whether or not the three 6/-4 pay offs are equal.

5.3 Comments

One observation about the stated algorithm is that if it sometimes played random moves during its exploitation phase and never terminated, then it would eventually find the truly optimal action. This would happen as the observed averages tend to the true averages with probability one over time. Also since the sample standard deviations tend towards zero over time then the algorithm as presented is guaranteed to eventually terminate at some stage.

Because this algorithm performs the exploitation deterministically based on information which is shared between all of the agents, it is guaranteed to achieve coordination at every stage and in particular in the choice of optimal action.

A question is whether it will output an optimal action when it terminates. There are several ways that it can fail

to do this:

- It can simply never observe the optimal joint action during the exploration phase.
- It can omit the optimal joint action from the set of potential optimal joint actions, either upon formation of this set or later during exploitation, due to misleading results in the expected payoff.

The first problem is difficult to avoid and the chance of it occurring tends to zero as ExplorationCutOff tends to infinity. In practice for a 3 x 3 grid an exploration cut off of 4 gives a $1 - (8/9)^{32} \approx 98\%$ chance of seeing a single optimal action. An exploration cut off of 6 suffices to make it very unlikely that the optimal action is missed during exploration.

It is unfortunate that larger grids require larger values of ExplorationCutOff than smaller ones to reach the same degrees of confidence as this goes against the overall design philosophy by requiring (very basic) knowledge of the games structure to decide on the inputs to use. However this seems unavoidable.

The second problem is also unavoidable, although a large value of any of ExplorationCutOff, SDTolerance or MinSD will make this a rare occurrence.

If ExplorationCutOff is high then the observed averages will be very likely to be close to the true averages and so the optimal action is unlikely to be omitted due to having a low observed payoff. If MinSD or SDTolerance are very high then the set of possibly optimal actions is likely to be very large and include the optimal action.

The stated algorithm does struggle with joint actions which produce bad payoffs most of the time but very rarely produce a hugely positive action. In general such problems are fundamentally difficult for any reinforcement learner.

There currently is no version of this algorithm for games with multiple states. In principle the main ideas are still valid in this case: just track the Q values and their sample standard deviations instead of the raw means. A problem arises when calculating the standard deviations of the Q values.

As the transition probabilities are also sample means they are approximately normally distributed and need their standard deviations taken into account. One then ends up with a distribution for the observed mean of each Q value formed by summing the results of taking the products of various normal distributions. This distribution is rather difficult to work with. Having said that, this would probably be possible and would probably produce useful results for multi-stage games.

6 Conclusions

We have shown on a wide range of difficult coordination games from the research literature that learning of coordination can be effectively achieved with very simple

and efficient approaches, where agents can observe each other's actions. We thus strongly argue for the use of the simple techniques as baselines in any future research on joint-action learning of coordination.

That such approaches are effective, and indeed close to optimal in terms of speed, in coordination games should not be surprising.

For a single stage game with constant payoffs the only thing a joint action learner has to worry about is viewing each possible joint action as quickly as possible. Since there is also in general in these games no correlation between the values of joint actions using most of the same actions, there is no way to predict good joint actions from the past results of observing other joint actions and random play is the best policy. Also the games under consideration are small enough to render observing the entire state/action space feasible. Multi-stage games with stochastic pay offs are a little more difficult but not at a fundamental level.

However in typical real world problems, you may only have time to observe a small fraction of the total state/action space and solutions which are at least locally optimal may well be clustered closely together. For such problems the simple learners presented in this paper will tend not to perform very well.

Acknowledgements

We thank Spiros Kapetanakis for many useful discussions on this research.

References

- Georgios Chalkiadakis and Craig Boutilier. Coordination in multiagent reinforcement learning: A bayesian approach. In *Proceedings of the Second international conference on Autonomous Agents and Multiagent Systems.*, 2003.
- Caroline Claus and Craig Boutilier. Dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998, 1998.
- C. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge University, 1989.
- Spiros Kapetanakis and Daniel Kudenko. Reinforcement learning of coordination in cooperative multiagent systems. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 2002.
- Michael L Littman. Markov games as a framework for multi - agent reinforcement learning. In *Proceedings of the Eleventh International Conference on machine Learning*, 1994.

Ann Nowe, Johan Parent, and Katja Veerbeck. Social agents playing a periodical policy. In *Proceedings of the 12th European Conference on Machine Learning*, 2001.

Xiaofeng Wang and Tuomas Sandholm. Reinforcement learning to play an optimal nash equilibrium in team markov games. In *Proceedings of the Sixteenth Conference on Neural Information Processing Systems.*, 2002.

Time Management Adaptability in Multi-Agent Systems

Alexander Helleboogh*

*AgentWise, DistriNet, Department of Computer Science K.U.Leuven, Belgium

Alexander.Helleboogh@cs.kuleuven.ac.be

Tom Holvoet*

Tom.Holvoet@cs.kuleuven.ac.be

Danny Weyns*

Danny.Weyns@cs.kuleuven.ac.be

Abstract

So far, the main focus of research on adaptability in multi-agent systems (MASs) has been on the agents' behavior, for example on developing new learning techniques and more flexible action selection mechanisms. However, we introduce a different type of adaptability in MASs, called *time management adaptability*. Time management adaptability focuses on adaptability in MASs with respect to execution control. First, time management adaptability allows a MAS to be adaptive with respect to its execution platform, anticipating arbitrary and varying timing delays which can violate correctness. Second, time management adaptability allows the execution policy of a MAS to be customized at will to suit the needs of a particular application. In this paper, we discuss the essential aspects of time management adaptability: (1) we introduce *time models* as a means to explicitly capture the execution policy derived from the application's execution requirements, (2) we classify and evaluate *time management mechanisms* which can be used to enforce time models, and (3) we describe a *MAS execution control platform* which combines both previous aspects to offer high level execution control.

1 Introduction and Motivation

Traditionally, the scope of research on adaptability in multi-agent systems (MASs) has been focused on trying to improve adaptability of individual agents and agent aggregations. As a consequence, the progress made by improving learning techniques and developing more flexible action selection mechanisms and interaction strategies over time is remarkable. This, however, may not prevent us from opening up our perspective and investigating other issues requiring adaptability in MASs. This paper is a first report on ongoing work and introduces *time management adaptability* as an important source of adaptability with respect to the execution control of MAS applications.

1.1 The Packet-World

We introduce the Packet-World application we have developed (Weyns and Holvoet, 2002), since this is used as an example MAS throughout the text.

The Packet-World consists of a number of different colored packets that are scattered over a rectangular grid. Agents that live in this virtual world have to collect those packets and bring them to their corresponding colored destination. The grid contains one destination for each color. Figure 1 shows an example of a Packet-World with size 10 wherein 5 agents are situated. Squares symbolize packets and circles are delivery points. The colored rings symbolize pheromone trails discussed below.

In the Packet-World, agents can interact with the envi-

ronment in a number of ways. We allow agents to perform a number of basic actions. First, an agent can make a step to one of the free neighbor fields around him. Second, if an agent is not carrying any packet, it can pick one up from one of its neighboring fields. Third, an agent can put down the packet it carries on one of the free neighboring fields around it, which could of course be the destination field of that particular packet.

It is important to notice that each agent of the Packet-World has only a limited view on the world. This view only covers a small part of the environment around the agent (see figure 1).

Furthermore, agents can interact with other agents too. We allow agents to communicate indirectly using stygmergy (Sauter et al.; Parunak and Brueckner, 2000; Brueckner, 2000): agents can deposit pheromone-like objects at the field they are located on. These pheromones evaporate over time and can be perceived by other agents. In the Packet-World, this allows agents to construct pheromone trails between clusters of packets and their destination (Steels, 1990). In figure 1 pheromones are symbolized by colored rings. The color of the ring corresponds to the packet color. The radius of the ring is a measure for the strength of the pheromone and decreases as the pheromone evaporates. Other agents noticing a pheromone trail can decide to follow it in the direction of increasing pheromone strength to get to a specific destination (e.g. when they are already carrying a packet of corresponding color). On the other hand, agents can also decide to follow the trail in the direction decreasing pheromone strength leading to the packet cluster (e.g.

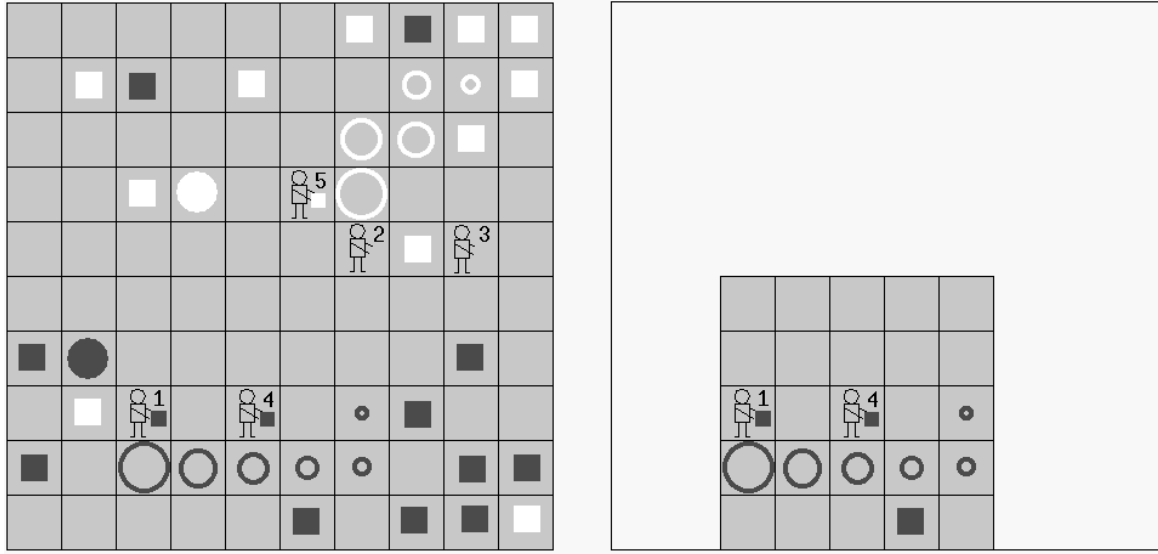


Figure 1: The Packet-World: global screenshot (left) and view range of agent nr.4

when they are not carrying anything). Hence they can help transporting the clustered packets, while reinforcing the evaporating pheromone trail on their way back to the destination. In this way stygmergy provides a means for coordination between the agents which goes beyond the limitations of the agents' locality in the environment.

1.2 Problem Statement

So far, time in MASs is generally dealt with in an implicit and ad hoc way: once the agents have been developed, they are typically hooked together using a particular activation regime or scheduling algorithm (see figure 2), without decent time management. MASs with an implicit notion of time are generally not adapted at all to run-time variations of timing delays introduced by the underlying execution platform, e.g. network delays, delays due to scheduling policies, etc. Moreover, these variations with respect to the execution of the agents can have a severe impact on the behavior of the MAS as a whole (Axtell, 2000; Page, 1997). The reason for this is that mostly the temporal relations existing in the problem domain differ significantly from the arbitrary and variable

time relations in an execution platform (Fujimoto, 1998), emphasizing the need for an explicit time management. In other words, delays in an execution platform are based on quantities which have nothing to do with the problem domain. To illustrate this, consider the following examples from our Packet-World application:

1. From the MAS's point of view, agents with simpler internal logic are expected to react faster than agents with more complex internal logic. However, consider a packet lying in between a cognitive agent and a significantly faster reactive agent, for instance the white packet between agents 2 and 3 in figure 1. In case both agents start reasoning at the same time, it seems obvious that the reactive agent can pick up the packet before the cognitive one can. However in practice, the response order of both agents is arbitrary, because the underlying execution platform could cause the cognitive agent's process to be scheduled first, allowing it to perform its reasoning and pick up the packet before the faster reactive agent even got a chance.
2. The agents can deposit pheromones drops in the environment to coordinate their activity. These pheromones evaporate over time. Because the effectiveness of pheromone-based communication is strongly dependent upon this evaporation rate, the latter is tuned to suit the needs of a particular application. However, fluctuations in the load of the underlying execution platform can cause agents to speed up or slow down accordingly, leading to a significant loss of pheromone effectiveness which affects the overall behavior of the MAS.
3. Problems can also arise with respect to the actions agents can perform. In our Packet-World applica-

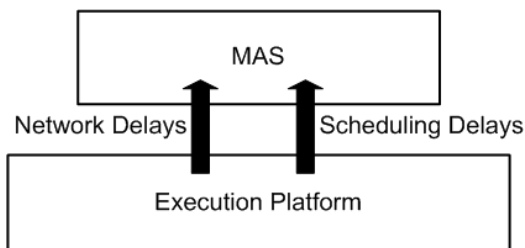


Figure 2: MAS directly built upon an execution platform

tion the time period it takes to perform a particular action must be the same for all agents. However, fluctuations in processor load can introduce variabilities with respect to the execution time of actions. As a consequence a particular action of an agent can take longer than the same action performed by other agents. This leads to agents arbitrarily obtaining privileges compared to other agents due to the execution platform, a property which is undesired in our problem domain.

The examples above show that a MAS without explicit time management is not adapted to varying delays introduced by the execution platform, which can be the cause of unforeseen or undesired effects. Hence the execution of all entities within a MAS has to be controlled according to the requirements at the level of the MAS, irrespective of execution platform delays. In this paper, we introduce *time management adaptability* as a generic solution to this problem and a more structured way to control the execution of a MAS.

1.3 Time Management Adaptability

Time management adaptability allows the execution of a MAS to be controlled by ensuring that all temporal relations which are essential from a conceptual point of view are correctly reproduced in the software system. Hence a particular execution policy for each MAS can be enforced. Second, time management adaptability allows easy adaptation of a MAS's execution, because it deals with time in an explicit manner and introduces execution control into a MAS as a separate concern. In order to achieve this, time management adaptability consists of three main aspects (see figure 3):

1. **Time models** are necessary to explicitly model the way time considered in the problem domain, without taking into account the underlying execution platform. Hence time models capture time at level of the MAS application. Time models are explicit which allows them to be easily adapted to reflect the custom needs of a MAS application.
2. **Time management mechanisms** are a means to ensure the consistency of time as being defined at the MAS level, even in the presence of arbitrary delays introduced by the execution platform.
3. A **MAS execution control platform** combines both time models and time management mechanisms to control the execution of a MAS. In a MAS execution control platform time models are used to capture the execution requirements and time management mechanisms are employed to prevent time models from being violated during execution. In this way the conceptual perception of time can be decoupled from the timing delays of the platform on which the MAS executes.

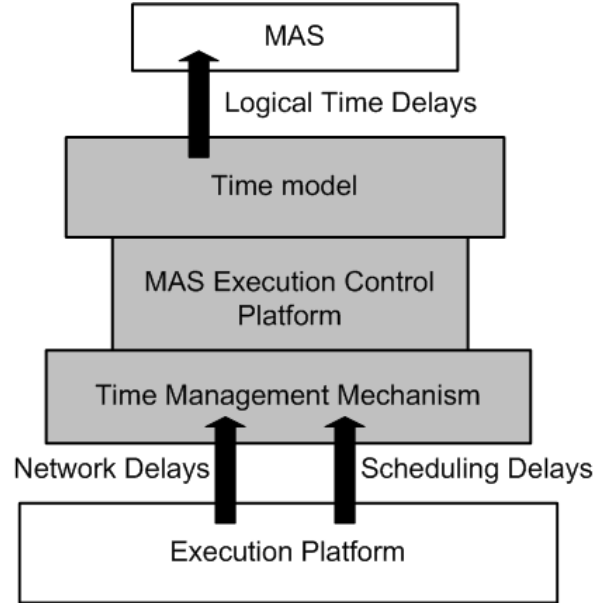


Figure 3: Time management adaptability in MASs

Outline of the paper. We first clarify the concept of time in MASs in section 2. We then discuss the various parts of time management adaptability: in section 3 we elaborate on *time models*. In section 4, the main *time management mechanisms* existing today are evaluated, and section 5 discusses *MAS execution control platforms* more in detail. Finally, we look forward to future work in section 6 and conclude in section 7.

2 Time in MAS

2.1 Causality And Time

Time in MASs is important because it determines causality (Schwarz and Mattern, 1994). To illustrate this, we start from the fundamental characteristics of MASs, as stated in the definition of Wooldridge and Jennings: “an agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives” (Wooldridge and Jennings, 1995). Agents are not isolated entities, but instead they are situated in an environment which they can perceive and on which they can act. By means of this shared environment, the actions of one agent have an influence on other agents. The order in which the actions take place in the environment determines the actual causality between agents.

We now elaborate on what causes a particular ordering of actions to arise and hence what actually determines causality in MASs. By means of coordination, agents can agree upon the order. However, to see what is determining the order in the absence of coordination, we return to the autonomy property of individual agents:

agents autonomously decide *when* to perform an action. Therefore in MASs, no global flow of control can be identified which unambiguously determines causality. Instead, each agent has its own, local flow of control. As a consequence, in the absence of explicit coordination between agents, the relative timing between their corresponding local control flows determines in which order their actions on the environment happen. We conclude that in MASs, time determines causal dependencies between non-coordinating agents.

2.2 Different Concepts of Time

One of the most common points of confusion is what is actually meant by time in software systems. We start from (Fujimoto, 1998) to distinguish 2 sorts of time which are of relevance for the rest of the paper:

- **Wallclock time** is the (execution) time as measured on a physical clock while running the software system. For example, in the Packet-World the execution of a particular agent to determine its next action might take 780 milliseconds on a specific processor.
- **Logical time** (also called virtual time) is the software representation of time as experienced in the problem context. For example, the current logical time of our application could be represented by an integer number; after executing the program for 37 minutes of wallclock time, 892 units of logical time may have passed.

According to the way logical time advances in a system, a number of execution modes can be distinguished (Fujimoto, 1998). In a *real-time execution*, logical time advances in synchrony with wallclock time. Here we are mainly concerned with *as fast-as-possible executions*, which attempt to advance logical time as quickly as possible, without direct relationship to wallclock time. As an example: for a software system it could be that after 5 minutes of wallclock time, 100 units of logical time may have been processed, while after 10 minutes of execution time, 287 logical time units have passed in an as fast as possible execution, instead of 200 for real-time execution.

3 Time Models

Time models are inspired by research in the distributed simulation community, where they are used implicitly to assign logical time stamps to all events occurring in the simulation (Lamport, 1978; Misra, 1986). In software simulations, the logical time stamp of an event corresponds to the physical time the event was observed in the real world which is being simulated.

However, we extend the use of time models from pure simulation contexts to execution control for MASs in general. Here, logical time is not used to obtain correspondence to physical time, which has no meaning outside the

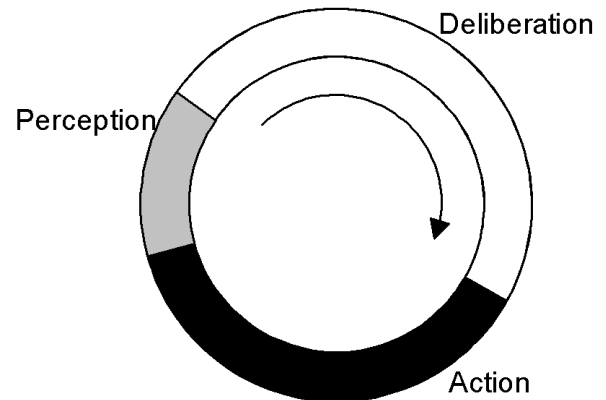


Figure 4: A typical agent control flow cycle

scope of simulation, but as a means to express causality in a MAS (see section 2.1). Also in contrast to software simulations, time models are now explicitly represented, which allows them to be easily adapted.

A time model captures the requirements with respect to time, as exposed in the problem context of most applications. More precisely, a time model defines how the duration of various activities in a MAS is related to logical time, and the order of activities in logical time is used as a means to express causality between the entities in a MAS. In this way a time model allows the developer to describe the causal relationships which are essential for the correct working of the MAS and hence must be ensured during the execution of the MAS on any particular platform.

Time models capture time relations at the level of the MAS's problem domain. A first important thing which needs to be done when considering time models, is investigating the structure of a MAS to identify all entities that need to be time modeled.

3.1 Time Modeling Agents

In the previous section we stated that time determines causal relations between agents, since each of the latter has its own control flow. Hence the first important MAS entities which require time modeling are the agents. Because generally agents can perform several activities, time modeling an agent requires assigning durations in logical time to all of its activities. For our discussion, we assume that an agent has a control flow cycle as the one depicted in figure 4.

3.1.1 Agent deliberation

A first important activity an agent can perform, is internal deliberation. The purpose of this activity is determining the next action the agent is going to perform. Depending on the context, agent's deliberation can be very simple (e.g. stimulus-response behavior in reactive agents) or

immensely complex (e.g. sophisticated learning and planning algorithms used in cognitive agents).

In the context of agent-based simulation, time modeling agent's deliberation has received a lot of interest. We evaluate various time models which have been proposed to describe how much logical time the deliberation of the agents takes, and discuss their relevance for execution control.

- A constant time model (Uhrmacher and Kullick, 2000) for the agents' deliberation implies that the deliberation of all agents is performed in a constant logical time, irrespective of the actual wallclock time that is needed to execute the deliberation. By assigning constant logical time durations to the deliberation activity of each agent, one can determine the relative speed of all agents within a MAS at conceptual level, irrespective of their implementation or execution efficiency.
- If the functionality of an agent has to be time sensitive, a model assuming that the agent's deliberation always happens in a constant logical duration is no longer suitable. In this case, the logical duration of the deliberation activity can be modeled as a function of *reasoning primitives* (Anderson, 1997), e.g. evaluating the results of a perception action, updating its internal world model, etc. In this case, deliberation time cannot be predicted a priori, because only those primitives which are actually used during a particular deliberation phase are taken into account to determine the duration in logical time. However, although this approach is conceptually feasible, it requires zooming in into the agent's internal working to monitor what the agent is actually deliberating about.
- A popular approach uses the actual computer instructions as a basis for determining the logical deliberation time. At first sight, modeling the logical deliberation time of an agent as a function of the number of code instructions executed during deliberation could be considered an extreme example of the previous approach which treats each computer instruction as a reasoning primitive (Anderson, 1997). However counting computer instructions is related to the underlying execution platform rather than to the conceptual level of the MAS. Hence precaution has to be taken when using it in a time model for execution control, since this approach can cause a number of unexpected effects to occur. Using this model, the programming language used for an agent, the efficiency of implementation and the presence of GUI or debug code have a significant influence on the logical deliberation time, although these aspects have no conceptual meaning. Another drawback of this approach is that a "timed" version of the language in which the agents are programmed is required, e.g. Timed Common Lisp.

- The logical deliberation time can be modeled as a function of the wallclock time used for executing the deliberation (Anderson, 1995). This approach is also not feasible from conceptual point of view, since the logical duration is now susceptible to the load and performance of the underlying computer system.

3.1.2 Agent Action

A second important activity of an agent is performing actions (see figure 4). Compared to agent deliberation, time modeling agents' actions on the environment has received little interest. However, in the context of execution control, imposing time models on the actions agents perform is indispensable. Depending on the problem context, actions can be assigned a particular cost expressed as a time penalty the agent receives for performing the action. From the agents' point of view, this period of logical time can be considered as the time the agent needs to complete that particular action. Since meanwhile the agent is not allowed to perform anything else, this approach can be used to model the frequency an agent is allowed to perform actions, irrespective of the underlying execution platform.

3.1.3 Agent Perception

Agent perception is also an agent activity which must be time modeled, however this is often neglected. Assigning a logical time duration to perceptual actions is analogous to time modeling other actions. An advantage of time modeling perception is for example that it can be a means to better control agents which continuously poll the environment by means of perception.

3.2 Time Modeling Ongoing activities

Besides the agents, there can be other entities within a MAS which require time modeling. In MASs, there is an increased environmental awareness. The environment itself is often dynamic and evolves over time. As a consequence causal relations do not only occur between interacting agents: the environment itself can contain a number of *ongoing activities* which are essential for the correct working of the MAS as a whole (Parunak et al., 2001). Ongoing activities are characterized by a state which evolves over time, even without agents affecting it. Agents can often initiate ongoing activities and influence their evolution. For example a ball in a robocup soccer game that was kicked by an agent, or the pheromones in our Packet-World application which evolve continuously. We conclude that the environment is not passive, but active, and responsible for the evolution of all ongoing activities.

In many MAS applications however, the dynamics of ongoing activities are dealt with in an ad hoc way. For example, the evaporation rate of pheromones is modeled in wallclock time, such that the correlation between agent activity on the one hand and pheromone activity

on the other hand is not guaranteed. As a consequence optimal coordination effectiveness can hardly be maintained: varying loads on the execution platform cause agent activity to slow down or speed up accordingly, while pheromone evaporation is determined upon wall-clock time and hence not adaptive to platform loads. Therefore it is very useful to provide ongoing activities of the environment with a time model, which allows their activity to be controlled to suit the needs of the MAS application.

3.3 Time Models: A Case

We now return to the Packet-World application, and illustrate the use of time models to capture the causal relations which are necessary for the correct working of the MAS and hence must be ensured on any particular platform. The problem statement (see section 1.2) mentions a number of typical problems with respect to execution control which arise in our Packet-World application. We now elaborate on describing the requirements in our problem examples to derive time models.

In the Packet-World, the following actions can be distinguished: a move action, a pick up packet action, a put packet down action and a drop pheromone action. Stated formally:

$E = \{move, pick, put, drop\}$
 with $move$ = move action
 $pick$ = pick up packet
 put = put down packet
 $drop$ = drop pheromone

With E the set of all possible actions on the environment.

In our application, there is only one perceptual action: agents can see their neighborhood. Formally:

$P = \{look\}$
 with $look$ = visual perception

With P the set of perceptual actions.

We distinguish two types of agents in our Packet-World: reactive agents and cognitive agents. Each agent is either reactive or cognitive. Stated formally:

$$A^R = \{a_1^r, a_2^r, \dots, a_n^r\}$$

$$A^C = \{a_1^c, a_2^c, \dots, a_m^c\}$$

$$A = A^R \cup A^C = \{a_1, a_2, \dots, a_{m+n}\}$$

With A^R the set of all reactive agents in our Packet-World application, A^C the set of all cognitive agents, and A the set of all agents, reactive and cognitive.

3.3.1 Action Requirements

We take a closer look at the third problem mentioned in section 1.2. In our Packet-World application it was observed that the underlying execution platform can have an arbitrary influence on the time it takes to perform an action. However, in the problem domain it is required the same amount of time is needed for all agents to perform a particular action. Stated formally:

$$\begin{aligned} \forall a_i \in A; move, pick, put, drop \in E : \\ \Delta T_{act}(move, a_i) &= cst_{move} \\ \Delta T_{act}(pick, a_i) &= cst_{pick} \\ \Delta T_{act}(put, a_i) &= cst_{put} \\ \Delta T_{act}(drop, a_i) &= cst_{drop} \\ cst_{move}, cst_{pick}, cst_{put}, cst_{drop} &\in \mathbb{N} \end{aligned}$$

With A the set of all agents, E the set of all actions on the environment, $\Delta T_{act}(e, a_i)$ the logical duration of action e performed by agent a_i , and \mathbb{N} the set of natural numbers.

The previous equations can also be expressed as:

$$\begin{aligned} \forall a_i \in A; \forall e \in E : \\ \Delta T_{act}(e, a_i) &= cst_e \\ cst_e &\in \mathbb{N} \end{aligned}$$

Since perception is considered as a kind of action in our application, we obtain the following expression:

$$\begin{aligned} \forall a_i \in A; look \in P : \\ \Delta T_{per}(look, a_i) &= cst_{look} \\ cst_{look} &\in \mathbb{N} \end{aligned}$$

With A the set of all agents, P the set of all perceptual actions, $\Delta T_{per}(look, a_i)$ the logical duration of perceptual action $look$ performed by agent a_i , and \mathbb{N} the set of natural numbers.

Stated more generally:

$$\begin{aligned} \forall a_i \in A; \forall p \in P : \\ \Delta T_{per}(p, a_i) &= cst_p \\ cst_p &\in \mathbb{N} \end{aligned}$$

3.3.2 Deliberation Requirements

We now return to the first example of our Packet-World application. The problem was the fact that the underlying execution platform can influence the reaction speed of the agents, leading to a response order which is arbitrary. However, this is not desired from a conceptual point of view, where we would like the reactive agent to always react faster than the cognitive agent, in case both start deliberating at the same time. Based on an agent's control flow cycle as depicted in figure 4, the moment in logical time an agent completes an action can be stated formally

as:

$$\forall a_i \in A; \forall e \in E; look \in P : \\ T_{end}(e, a_i) = T_0 + \Delta T_{per}(look, a_i) + \Delta T_{delib}(a_i) + \Delta T_{act}(e, a_i)$$

With $T_{end}(e, a_i)$ the logical time the action e of agent a_i completes, T_0 the logical time that a new cycle in the control flow of agent a_i starts, $\Delta T_{per}(look, a_i)$ the logical duration of the perception $look$ performed by agent a_i , $\Delta T_{delib}(a_i)$ the logical duration of the deliberation of agent a_i , and $\Delta T_{act}(e, a_i)$ the logical duration of action e performed by agent a_i .

The requirement that reactive agent can always pick up the packet before cognitive agent in case both start deliberating at the same time, is hence formalized as follows:

$$\forall a_i^r \in A^R; \forall a_j^c \in A^C; pick \in E : \\ T_{end}(pick, a_i^r) < T_{end}(pick, a_j^c)$$

With A^R the set of reactive agents, A^C the set of cognitive agents, and E the set of all actions on the environment.

By substitution we obtain:

$$T_0 + cst_{look} + \Delta T_{delib}(a_i^r) + cst_{pick} < T_0 + cst_{look} + \Delta T_{delib}(a_j^c) + cst_{pick}$$

Simplifying both sides of the equation gives us:

$$\Delta T_{delib}(a_i^r) < \Delta T_{delib}(a_j^c)$$

With $a_i^r \in A^R$ a reactive agent, and $a_j^c \in A^C$ a cognitive agent.

Hence to assure that the reactive agent always acts faster than the cognitive one, the logical duration of the agents' deliberation needs to be modeled such that reactive agent's deliberation duration is smaller than the cognitive agent's deliberation time.

3.3.3 Pheromone Requirements

Finally, we take a closer look at the second example. The load on the underlying execution platform causes the agents' execution speed to change accordingly. However, if we want to maintain pheromone effectiveness, we need a continuous adaptation of the pheromone evaporation rate to the agents' execution speed. Hence from conceptual point of view, we want to relate pheromone activity to agent activity. As a first step, we use a simplistic model for pheromone activity. For the pheromone evaporation rate in our Packet-World application we can state more formally:

$$\Delta T_{evap} = cst_{evap} \\ cst_{evap} \in \mathbb{N}$$

with ΔT_{evap} the logical duration it takes for a pheromone to evaporate until only half of its initial strength is remaining. Agent activity is related to logical time. Relating pheromone activity to the same logical clock, instead of the wallclock, allows the dynamics of both agents and pheromones to be coupled.

3.4 Deriving a Time Model

In section 3.3 we formulated a number of requirements which have to be met to allow the execution of our MAS to evolve appropriately. To obtain a time model, one has to assign specific values to the various activities, which express the logical durations. These values are expressed in *logical time units* (LTU). The requirements mentioned above give rise to an array of constraints which all have to be satisfied to come to a suitable time model for the application.

A possible time model for our Packet-World application is given:

$$\forall a_i \in A : \\ \Delta T_{act}(move, a_i) = 3 \text{ LTU} \\ \Delta T_{act}(pick, a_i) = 2 \text{ LTU} \\ \Delta T_{act}(put, a_i) = 2 \text{ LTU} \\ \Delta T_{act}(drop, a_i) = 1 \text{ LTU} \\ \Delta T_{per}(look, a_i) = 1 \text{ LTU}$$

$$\forall a_i^r \in A^R : \\ \Delta T_{delib}(a_i^r) = 4 \text{ LTU}$$

$$\forall a_j^c \in A^C : \\ \Delta T_{delib}(a_j^c) = 12 \text{ LTU}$$

$$\Delta T_{evap} = 100 \text{ LTU}$$

Note that this time model is only an example which satisfies all requirements of the application. The values assigned to the various activities indicate units of logical time. Note also that these values can be altered, as long as the constraints expressing the requirements of the Packet-World application are not violated. Hence various execution policies can be experimented with.

4 Time Management Mechanisms

By describing time models the developer imposes an order on MAS activities, dictated by logical time. However, as illustrated in the introduction, the temporal characteristics of the execution platform are not necessarily the same as those described by the logical time model. As a consequence, we additionally need *time management mechanisms* which enforce the time models and hence *preserve causality*. This means avoiding that any event with a logical time in the future can have influence on things with a

logical time in the past, even in the presence of arbitrary network delays or computer loads.

Distributed and agent-based simulation communities have been investigating the consistency of logical time in simulations for a long time. All events happening are ordered and hence causally related by means of the global notion of logical time. Therefore various time management mechanisms have been developed to prevent causality errors:

- **Execution directed by clock.** In this approach the logical time of the system is discretized in a number of intervals of equal size. The interval size is called *time-step*. Global synchronization schemes force all entities to advance together in a lock-step mode, and hence the execution of the system proceeds synchronously. In the case of MASs, a drawback is that synchronous execution forces all agents to act at the pace of the slowest one, which severely limits execution speed (Weyns and Holvoet, 2003). Moreover, since a central authority must control and keep track of the execution of all agents in the system, the cost of synchronous approaches increases rapidly as the number of agents grows.
- **Execution directed by events.** In this case, events are generated by all entities (Lamport, 1978), and each event has a precise logical time stamp which allows sorting them. During execution, the next event to be processed is the one with the smallest logical timestamp, ensuring causality and thereby skipping periods of inactivity. However in a distributed context (distributed discrete event simulation (Misra, 1986)), a system is modeled as a group of communicating entities, referred to as logical processes (or LPs). Each LP contains its own logical clock (indicating its local logical time) and all LPs process events asynchronously and advance at different rates, which allow a significant speedup, but may cause causality errors. Hence, for asynchronous execution additional synchronization is needed to ensure that each LP processes messages in increasing logical time order:
 - **Conservative synchronization.** In conservative synchronization (Chandy and Misra, 1981) each LP only processes events when it can guarantee that no causality errors (out of (logical) time order messages) will occur. This causes some LPs to block, possibly leading to deadlock. The performance of conservative synchronization techniques relies heavily on the concept of lookahead, but the autonomous, proactive behavior of agents could severely restrict the ability to predict events (Uhrmacher and Gugler, 2000). Moreover, to determine whether it is safe for an agent to process an event, information about all other agents must

be taken into account, limiting the scalability of this approach.

- **Optimistic synchronization.** In optimistic approaches, causality errors are allowed, but some roll-back mechanism to recover from causality violations is defined (e.g. time warp (Jefferson and Sowizral, 1985)). In the case of MASs, however, the cost imposed by the roll-back mechanisms can easily outweigh the benefits (Uhrmacher and Gugler, 2000), and increases rapidly as the number of agents grows.

5 MAS execution control platform

To control the execution of a MAS in an appropriate way, a *MAS execution control platform* must provide support for both explicit time models on the one hand and time management mechanisms on the other hand.

First, logical time models are needed as a means for the developer to explicitly express an execution policy for all MAS activities. However, the execution policy expressed in the time model has to be enforced at implementation level, irrespective of delays in the underlying execution platform. For this reason, time management mechanisms are needed. They prevent time models from being violated, and ensure the execution of the MAS behaves according to the policy which is described.

As a consequence, a MAS execution control platform must provide an integrated support for both time models and time management mechanisms to achieve the advantages below:

- **Higher level of abstraction.** The delays of the underlying execution platform are decoupled from the MAS. From a developer's point of view, only the logical durations described in the time model apply and determine causality. As a consequence abstraction can be made of the execution platform delays.
- **Separation of concerns.** MAS entities can be developed without taking into account the policy which will be used for their execution. Controlling the execution of a MAS can now be considered as a separate concern. This relieves the developer from explicitly building execution control mechanisms and hard-coding them into the agents. The MAS execution control platform allows a time model representing the execution policy of the MAS to be described independently and enforced transparently.
- **Adaptability with respect to the execution platform.** By combining a logical time model and a time management mechanism to enforce it, the execution of a MAS is no longer affected by timing issues introduced by the execution platform. This results in MASs being adaptive with respect to the timing characteristics of the execution platform, an important

type of adaptability which is not often considered in the context of adaptive MASs.

- **Adaptability of the execution policy.** The explicit representation of the time model of a MAS allows the execution policy to be easily adapted. This enables fine-tuning of the existing execution policy and integration of new execution requirements.

6 Future Work

This paper is a first report on ongoing work investigating a generic and structured way to deal with execution control in the context of MASs. The approach was described in general, and a lot of work still needs to be done on various aspects described in this paper:

- We are currently working to improve the formalism for describing logical time models, to come to an approach which is generally applicable and more theoretically founded. Also in the context of time models, the dynamics of pheromones still need to be investigated more in depth.
- As shown in section 4, mechanisms enforcing time models and ensuring global causality are limited in scalability, making these approaches inefficient for use in a large-scale distributed MASs. A possible alternative presumes we abandon the notion of a global (logical) clock to determine causal relationships. Hence not all parts of the MAS are related in time, and there is only a locally shared notion of time. This follows from the observation that in a lot of MASs, agents only perceive and act locally. Based on this, it makes sense only to ensure temporal relationships between agents residing in each other's neighborhood, without modeling causality between agents far away from each other, at the benefit of increased scalability. Regional synchronization (Weyns and Holvoet, 2003) provides a flexible mechanism to dynamically detect clusters of agents, based on the overlap of so called *spheres of influence*. Within such clusters of agents, the mechanisms discussed in section 4 could be applied locally, hence avoiding scalability limitations at the cost of a loss of a global notion of logical time.
- Although the first results of our Packet-World case study are promising, other examples need to be investigated, and the design of a generic MAS execution control platform allowing a full separation of concerns is a major challenge which has to be tackled.

7 Conclusion

In this paper, we emphasized time management adaptability as a type of adaptability which has significant impor-

tance, although this type of adaptability is not often considered in the context of adaptive MASs. Time management adaptability is based on the important aspect of a logical *time model* to explicitly capture the execution policy which is essential for the correct functioning of the MAS. *Time management mechanisms* form a second important aspect of time management adaptability: they are needed to enforce time models. Time models and time management mechanisms are combined in *MAS execution control platforms*. As a consequence, the advantage of time management adaptability is twofold:

First, adaptability of the MAS with respect to the execution platform is considered: the combination of time models and time management mechanisms allows all essential causal relationships to be no longer affected by timing issues introduced by the execution platform. Hence causality in the MAS can remain invariant under various execution conditions.

Second, time management adaptability allows the execution policy to be adapted to suit the needs of the MAS application, providing a higher level of abstraction to deal with time in a MAS, and a means to introduce execution control as a separate concern.

References

- S. Anderson. *A Simulation Substrate for Real-Time Planning*. PhD thesis, 1995.
- Scott D. Anderson. Simulation of multiple time-pressured agents. In *Winter Simulation Conference*, pages 397–404, 1997.
- Robert Axtell. Effects of interaction topology and activation regime in several multi-agent systems. In *MABS*, pages 33–48, 2000.
- Sven A. Brueckner. *Return From The Ant - Synthetic Ecosystems For Manufacturing Control*. PhD thesis, Humboldt University Berlin, Department of Computer Science, 2000.
- K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(11):198–205, November 1981.
- R. Fujimoto. Time management in the high level architecture. *Simulation, Special Issue on High Level Architecture*, 71(6):388–400, 1998.
- David Jefferson and H. Sowizral. Fast concurrent simulation using the time warp mechanism. In *Proceedings of the SCS Multiconference on Distributed simulation*, pages 63–69, January 1985.
- Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

- J. Misra. Distributed discrete-event simulation. *Computing Surveys*, 18(1):39–65, March 1986.
- S. Page. On incentives and updating in agent based models. *Journal of Computational Economics*, 10:67–87, 1997.
- H. Van Dyke Parunak and Sven Brueckner. Ant-like missionaries and cannibals: Synthetic pheromones for distributed motion control. In Carles Sierra, Maria Gini, and Jeffrey S. Rosenschein, editors, *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 467–474, Barcelona, Catalonia, Spain, 2000. ACM Press.
- H. Van Dyke Parunak, Sven Brueckner, John Sauter, and Robert S. Matthews. Distinguishing environmental and agent dynamics: A case study in abstraction and alternate modeling technologies. *Lecture Notes in Computer Science*, 1972:19–??, 2001.
- John A. Sauter, Robert Matthews, and H. Van Dyke. Evolving adaptive pheromone path planning mechanisms.
- Reinhard Schwarz and Friedemann Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing*, 7(3): 149–174, 1994.
- Luc Steels. Cooperation between distributed agents through self-organization. *Decentralized A.I.*, 1990.
- A. Uhrmacher and K. Gugler. Distributed, parallel simulation of multiple, deliberative agents. In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS'2000)*, Bologna, pages 101–110. IEEE, May 2000.
- A. Uhrmacher and B. Kullick. Plug and test software agents in virtual environments. In *Winter Simulation Conference - WSC'2000*, 2000.
- D. Weyns and T. Holvoet. The packet-world as a case to study sociality in multi-agent systems. In *Autonomous Agents and Multi-Agent Systems, AAMAS 2002, Bologna, Italy*, 2002.
- D. Weyns and T. Holvoet. Regional synchronization for simultaneous actions in situated multiagent systems. *Multi-Agent Systems and Applications III, Lecture Notes in Computer Science*, LNAI 2691:497–511, 2003.
- Michael J. Wooldridge and Nicholas R. Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2):115–152, June 1995.

Reinforcement learning of coordination in heterogeneous cooperative multi-agent systems

Spiros Kapetanakis and Daniel Kudenko
{spiros, kudenko}@cs.york.ac.uk
Department of Computer Science
University of York, UK

Abstract

Most approaches to the learning of coordination in multi-agent systems (MAS) to date require all agents to use the same learning algorithm with similar (or even the same) parameter settings. In today's open networks and high inter-connectivity such an assumption becomes increasingly unrealistic. Developers are starting to have less control over the agents that join the system and the learning algorithms they employ. This makes effective coordination and good learning performance extremely difficult to achieve, especially in the absence of learning agent standards. In this paper we investigate the problem of learning to coordinate with heterogeneous agents. We show that an agent employing the FMQ algorithm, a recently developed multi-agent learning method, has the ability to converge towards the optimal joint action when teamed-up with one or more simple Q-learners. Specifically, we show such convergence in scenarios where simple Q-learners alone are unable to converge towards an optimum. Our results show that system designers may improve learning and coordination performance by adding a "smart" agent to the MAS.

1 Introduction

Learning to coordinate in cooperative multi-agent systems is a central and widely studied problem, see, for example, any of the following published work by Verbeeck et al. (2003), Lauer and Riedmiller (2000), Boutilier (1999), Claus and Boutilier (1998), Sen and Sekaran (1998). In this context, coordination is defined as *the ability of two or more agents to jointly reach a consensus over which actions to perform in an environment*.

To date, learning techniques require the multi-agent system to be *homogeneous*, i.e. all agents need to employ the same learning algorithm and often use the same (or at least similar) parameter settings to achieve optimal coordination.

In today's open networking environment the assumption of agent homogeneity is becoming increasingly unrealistic. Agents are designed by different individuals with different preferences and learning agent standards are virtually non-existent. This poses a problem to designers of open multi-agent systems who don't have control over the algorithms that the agents acting in these systems actually use. How can the designers ensure optimal coordination in the multi-agent system under such conditions?

In this paper we suggest that it may be possible to control learning performance in a heterogeneous multi-agent system by adding a specific agent to the population. More precisely, we investigate the applicability of the FMQ technique (Kapetanakis and Kudenko, 2002) for the reinforcement learning of coordination. We show that a team consisting of an FMQ agent and one or more sim-

ple Q-learners can achieve high probabilities of convergence to an optimal joint action in single-stage cooperative games, even in cases where Q-learners alone are unable to achieve any reasonable rates of convergence to the optimum. In other words, the FMQ-learner is able to "push" the simple Q-learner(s) to the optimum.

Note that the FMQ technique has been developed for *independent* agents that do not communicate or observe one another's actions, which is a more general and often more realistic assumption. This generality distinguishes our approach from alternatives such as the work by Wang and Sandholm (2002) and Chalkiadakis and Boutilier (2003).

This paper is structured as follows: we first present a common testbed for the study of learning coordination in cooperative multi-agent systems, namely single-stage cooperative games. We then introduce three particularly difficult examples of such games that we will use in the experiments. Following this, we present the experimental setup and discuss the results. We finish the paper with an outlook on future work.

2 Single-stage cooperative games

Markov games are a widely used testbed for studying reinforcement learning in multi-agent systems (Fudenberg and Levine, 1998; Littman, 1994). One particular variation of them which is often used in the study of coordination in multi-agent systems is that of single-stage cooperative games. In these games, the agents have com-

mon interests i.e. they are rewarded based on their joint action and all agents receive the same reward. In each round of the game, every agent chooses an action. These actions are executed simultaneously and the reward that corresponds to the joint action is broadcast to all agents at the same time.

A more rigorous account of single-stage cooperative games was given by Claus and Boutilier (1998). In brief, we assume a group of n agents $\alpha_1, \alpha_2, \dots, \alpha_n$ each of which has a finite set of *individual actions* A_i which is known as the agent's *action space*. In each iteration of the game, each agent α_i chooses an individual action from its action space to perform. The action choices of all agents put together make up a *joint action*, upon execution of which, all agents receive the reward that corresponds to the chosen joint action.

An example of such a game is the *climbing game* which was introduced by Claus and Boutilier (1998). This game, which is shown in Table 1, is played between 2 agents, each of which has 3 actions. If agent 1 executes action c and agent 2 executes action b , the reward they receive is 6. Obviously, the optimal joint action in this game is (a, a) as it is associated with the highest reward of 11.

	a	b	c
a	11	-30	0
b	-30	7	6
c	0	0	5

Table 1: The climbing game.

Our goal is to enable the agents to learn optimal coordination from repeated trials. To achieve this goal, one can use either *independent* or *joint-action* learners. The difference between the two types lies in the amount of information they can perceive in the game. Although both types of learners can perceive the reward that they receive for the execution of a joint action, the former are unaware of the existence of other agents whereas the latter can also perceive the actions of others. In this way, joint-action learners can maintain a model of the strategy of other agents and choose their actions based on the other participants' perceived strategies. In contrast, independent learners must estimate the value of their individual actions based solely on the rewards that they receive for their actions. In this paper, we focus on individual learners, these being more universally applicable.

In our present study, we focus on three particularly difficult coordination problems, the *climbing game* (Table 1), the *penalty game* (Table 2) and the *number-matching game* (Table 3). All three games are played between two agents. We also introduce general versions of the penalty and number-matching game which any number of agents can take part in. We use these to evaluate the applicability of this work on teams of more than 2 agents.

In the climbing game, it is difficult for the agents to converge to the optimal joint action (a, a) because of the

negative reward in the case of miscoordination. Incorporating this reward into the learning process can be so detrimental that both agents tend to avoid playing their respective components of the optimal joint action again. In contrast, when choosing action c , miscoordination is not punished so severely. Therefore, in most cases, both agents are easily tempted by action c .

Another way to make coordination more elusive is by including multiple optimal joint actions. This is precisely what happens in the penalty game. In this game, it is not only important to avoid the miscoordination penalties associated with actions (c, a) and (a, c) but it is equally important to agree on which optimal joint action to choose out of (a, a) and (c, c) . If agent 1 plays a expecting agent 2 to also play a so they can receive the maximum reward of 10 but agent 2 plays c (perhaps expecting agent 1 to play c so that, again, they receive the maximum reward of 10) then the resulting penalty can be very detrimental to both agents' learning process. In this game, b is the "safe" action for both agents since playing b is guaranteed to result in a non-negative reward, regardless of what the other agent plays.

	a	b	c
a	10	0	k
b	0	2	0
c	k	0	10

Table 2: The penalty game

In the last testbed, the number-matching game, the two agents can only receive a positive reward for playing the *same* action. Any of (a, a) , (b, b) or (c, c) will result in a positive reward with (c, c) being the optimal joint action. The difficulty, however, in solving this game stems from the fact that actions with a higher reward carry the risk of a higher penalty in the case of miscoordination. Every time the two agents play different actions, they are *both* punished with a penalty that matches the action of the more ambitious of the two. For example, if the agents play joint action (c, b) they will both receive a payoff of -3 because agent 1 tried its individual component of the optimal joint action.

	a	b	c
a	1	-2	-3
b	-2	2	-3
c	-3	-3	3

Table 3: The number-matching game

3 Reinforcement learning of Coordination

A popular technique for learning coordination in cooperative single-stage games is one-step Q-learning, a reinforcement learning technique. Since the agents in a single-stage game are stateless, we need a simple reformulation of the general Q-learning algorithm such as the one used by Claus and Boutilier (1998). Each agent maintains a Q value for each of its actions. The value $Q(\text{action})$ provides an estimate of the usefulness of performing this action in the next iteration of the game and these values are updated after each step of the game according to the reward received for the action. We apply Q-learning with the following update function:

$$Q(\text{action}) \leftarrow Q(\text{action}) + \gamma(r - Q(\text{action}))$$

where γ is the learning rate ($0 < \gamma < 1$) and r is the reward that corresponds to choosing this action.

In a single-agent learning scenario, Q-learning is guaranteed to converge to the optimal action independent of the action selection strategy. In other words, given the assumption of a stationary reward function, single-agent Q-learning will (eventually) converge to the optimal policy for the problem. However, in a multi-agent setting, the action selection strategy becomes crucial for convergence to *any* joint action. In fact, two regular Q-learners fail to converge to the optimal joint action in all games presented in the previous section.

In previous work (Kapetanakis and Kudenko, 2002), we developed a novel action selection heuristic, called FMQ. Using this technique, agents are able to converge to the optimal action in the three games from Section 2. FMQ is based on the Boltzmann strategy (Kaelbling et al., 1996) which states that agent α_i chooses an action to perform in the next iteration of the game with a probability that is based on its current estimate of the usefulness of that action, denoted by $EV(\text{action})$ ¹:

$$P(\text{action}) = \frac{e^{\frac{EV(\text{action})}{T}}}{\sum_{\text{action}' \in A_i} e^{\frac{EV(\text{action}')}{T}}}$$

In the case of Q-learning, the agent's estimate of the usefulness of an action may be given by the Q values themselves, an approach that has been usually taken to date. Instead, the FMQ approach uses the following formula to compute $EV(\alpha)$:

$$EV(\alpha) = Q(\alpha) + c * \text{freq}(\text{maxR}(\alpha)) * \text{maxR}(\alpha)$$

where:

- ① $\text{maxR}(\alpha)$ denotes the maximum reward encountered *so far* for choosing action α .
- ② $\text{freq}(\text{maxR}(\alpha))$ is the fraction of times that $\text{maxR}(\alpha)$ has been received as a reward for action α over the times that action α has been executed.
- ③ c is a weight that controls the importance of the FMQ heuristic in the action selection.

Informally, the FMQ heuristic carries the information of how frequently an action produces its maximum corresponding reward. Note that, for an agent to receive the maximum reward corresponding to one of its actions, the other agent must be playing the game accordingly.

4 Experimental results with 2 learners

This section contains our experimental results with a pair of heterogeneous reinforcement learners. We show that one FMQ-learner is indeed sufficient to achieve a high probability of convergence towards an optimal joint action when paired with a simple Q-learner. The simple Q-learner uses one-step Q-learning with the same temperature function as the FMQ-learner. We vary the learning rate γ for the Q-learner to show the performance of the FMQ/Q pair with different degrees of heterogeneity. The evaluation of the two approaches is performed on the climbing game, the penalty game and the number-matching game.

In all sections, we compare the performance of the FMQ/Q pair of learners with the baseline experiment of two homogeneous Q-learners using Boltzmann exploration with the following temperature function:

$$T(x) = e^{-sx} * \text{max_temp} + 1$$

where x is the number of iterations of the game so far, s is the parameter that controls the rate of exponential decay and max_temp is the value of the temperature at the beginning of the experiment. For a given length of the experiment (max_moves) and initial temperature (max_temp), the appropriate rate of decay (s) is automatically derived. Varying the parameters of the temperature function allows a detailed specification of the temperature. The settings for the baseline experiments are: $\text{max_temp} = 499, \gamma = 0.9$. All sets of experiments have been run 1000 times to minimise the variance in the results.

4.1 Evaluation on the Climbing Game

The climbing game has one optimal joint action, (a, a) , and two heavily penalised actions, (a, b) and (b, a) . In the evaluation that follows, we use the setting $\text{max_temp} =$

¹Kaelbling et al. (1996) introduce the estimated value as *expected reward* (ER).

499 and set the learning rate for the FMQ-learner to 0.9 and the confidence parameter of the FMQ-learner to $c = 10$. We show results for two experiment lengths, namely 1000 and 2000 moves. For each experiment length, we vary the learning rate for the standard Q-learner from 0.3 to 0.9. Figure 1 depicts the likelihood of convergence to the optimal joint action in the climbing game.

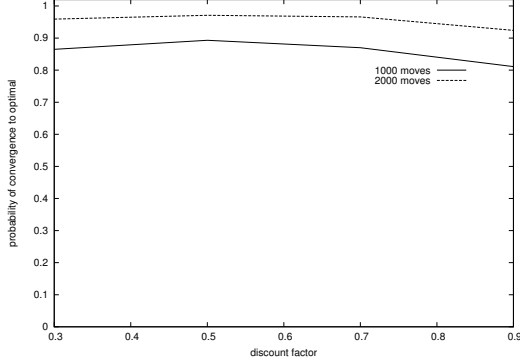


Figure 1: Probability of convergence to the optimal joint action in the climbing game.

The results shown in Figure 1 are significantly better than the baseline results. The two Q-learners with the baseline settings as explained above, only manage to converge to the optimal joint action with probability 0.168 in 1000 moves and with probability 0.19 in 2000 moves. This represents a major change in behaviour when one of the original Q-learners is substituted with an FMQ-learner.

4.2 Evaluation on the Penalty Game

The penalty game is harder to analyse than the climbing game because it has two optimal joint actions (a, a) and (c, c) for all values of k . The extent to which the optimal joint actions are reached by the agents is affected severely by the size of the penalty. However, the performance of the agents depends not only on the size of the penalty k but also on whether the agents manage to agree on which optimal joint action to choose. Figure 2 depicts the performance of the FMQ/Q pair of learners in the penalty game. Again, we set $\text{max_temp} = 499$ and set the learning rate for the FMQ-learner to 0.9. The confidence parameter of the FMQ-learner was set to $c = 10$ and we varied the Q-learner's learning rate from 0.3 to 0.9. In the interest of clarity, we show results for only one experiment length, namely 1000 moves.

The performance of the two Q-learners in the baseline experiment is slightly better in the penalty game than in the climbing game. For 1000 moves and the default settings, the two Q-learners' probability of convergence to either optimal action is shown in Table 4.

From Figure 2, it is clear that in the pair of agents solving the penalty game, the substitution of a Q-learner by

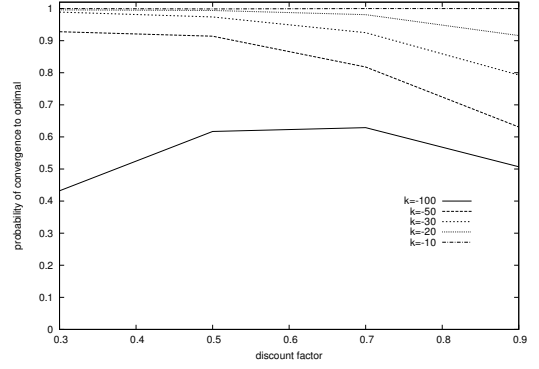


Figure 2: Probability of convergence to the optimal joint action in the penalty game.

an FMQ-learner is always beneficial, for all values of γ for the remaining Q-learner. The benefit is most striking when k is low and the Q-learner's learning rate is also low.

4.3 Evaluation on the Number Matching Game

The number-matching game turned out to be much less difficult than originally expected. The reason for this is that the FMQ-learner is not easily convinced that its c action is not a component of the optimal joint action when confronted with the reward of -3 for miscoordination. When the FMQ-learner witnesses a successful coordination on (c, c) and the resulting reward of 3 for playing c , this reward becomes the maximum reward corresponding to action c . This event, a successful coordination on the optimal joint action, inevitably happens in the early part of the experiment when the temperature is still relatively high and the agents play actions almost randomly. Eventually, the FMQ-learner manages to convince the standard Q-learner that (c, c) is the optimal joint action as it persists in playing c .

The results for the number-matching game are included in Figure 3. The settings for these results are: $\text{max_temp} = 499$, the learning rate for the FMQ-learner is 0.9 and the confidence parameter of the FMQ-learner is set to two values, namely 2 and 5. Normally, a value of $c = 2$ for the confidence parameter is too low to help the learners to converge to the optimal joint action. However,

k	(a, a)	(c, c)	total
-10	0.373	0.359	0.732
-20	0.256	0.276	0.532
-30	0.242	0.230	0.472
-50	0.194	0.222	0.416
-100	0.176	0.211	0.387

Table 4: Probability of convergence to optimal in the penalty game for the baseline experiment.

for the number-matching game where the optimal joint action has very low corresponding reward and the maximum penalty is not significantly greater in absolute value than the reward for the optimal joint action, even a value of 2 is enough to solve the game adequately. For illustration, Figure 3 depicts the performance of the learners for an experiment that is 1000 moves long, for both $c = 2$ and $c = 5$ for the FMQ-learner.

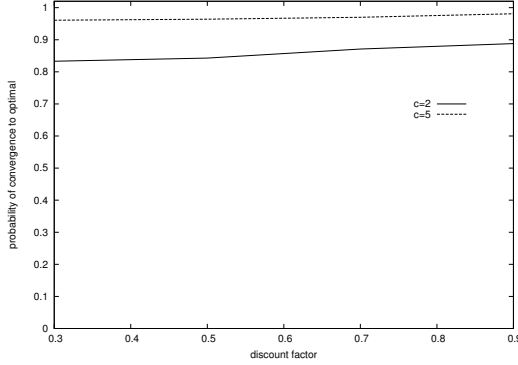


Figure 3: Probability of convergence to the optimal joint action in the number-matching game.

Again, the baseline experiment using two regular Q-learners is heavily outperformed by the FMQ/Q pair’s performance. In 1000 moves, the probability of convergence to the optimal action (c, c) in the baseline experiment is 0.32, which is significantly lower than the probability of success for the FMQ/Q pair, which is always above 0.833, for all tested values of γ and even for a low confidence value $c = 2$.

5 Evaluation with greater size teams

In this section, we present our experimental results with one FMQ-learner that is teamed up with more than one Q-learner. The results show that the FMQ-learner is still able to increase the probability of convergence to the optimum but not as drastically as in the two-agent case.

Two of the games from the above experiments can be generalised to more than 2 agents while keeping the overall philosophy of the game, namely the number-matching game and the penalty game. Their definitions are as follows:

The general number-matching game: if all agents play the same action, return the corresponding reward e.g. for action (a, a, \dots, a) return 1. If they play different actions, return the penalty corresponding to the agent that played the most ambitiously, e.g. for action (a, a, \dots, a, c) return -3.

The general penalty game: if all agents play a return 10, if all agents play c return 10, if all agents play b return 2. If any number of agents play a (or c) and at least one agent plays c (or a) return the penalty k as the group have just miscoordinated. For any other joint action, return a reward of 0.

In the sections that follow, we will use the short-hand notation $\langle \alpha \rangle$ to denote the joint action that results from all agents playing their individual action components α . For example, in the 4 agent case, $\langle b \rangle$ corresponds to the joint action (b, b, b, b) .

From the definitions of the two general games, we can see that they correspond appropriately to their original counterparts. The climbing game cannot be generalised to more than 2 agents as there is no symmetry to exploit in doing so. For that reason, we will perform the evaluation on agent teams of greater size than 2 only on the general penalty game and number-matching game.

In the text that follows, we will illustrate the performance of agent teams that comprise a single FMQ-learner and 2, 3 or 4 Q-learners. In all experiments, the agents use the same temperature function as previously with $\text{max_temp} = 499$ and $\gamma = 0.9$, for all agents. The FMQ-learner’s confidence parameter has been set to $c = 10$ throughout.

5.1 Evaluation with the Penalty Game

The general penalty game for teams of more than 2 agents is quite a challenging game. This is because even the FMQ heuristic that was so successful in two-agent experiments can be misled by greater size teams. For the heuristic to be more useful, the participating agents should all base their action selection decisions on the same reasons, namely that the action they are most tempted by is the one that produces better reward more often. In fact, had this been the case, i.e. if we were interested in the performance of a homogeneous FMQ-learning team, the probability of convergence to the optimal joint action would be significantly higher. For example, a team of 4 FMQ-learners would solve the general penalty game with probability comfortably greater than 0.95 in 5000 moves.

However, when one teams up more than 1 Q-learner with an FMQ-learner, the performance of the learning team suffers as the Q-learners play too randomly. This means that the frequency of getting high reward for coordinated actions is too low and the FMQ-learner is misled into believing that its b action is better than either a or c . In effect, although the FMQ-learner is still “pushing” the group towards some joint action, that joint action is (b, b) and not one of the optimal joint actions.

From Figure 4, we can see that although the experiments with 2 Q-learners and 1 FMQ-learner are still quite successful, those with greater size teams are not. This is attributed to the problem described above where the optimal joint action is simply not experienced often enough

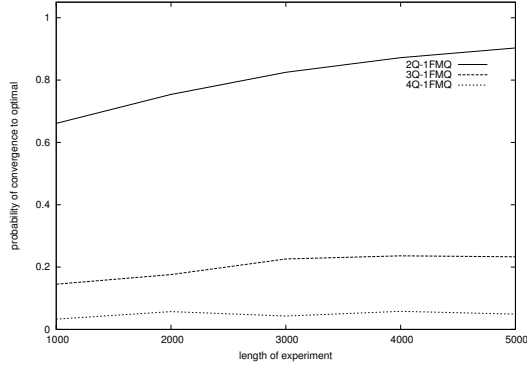


Figure 4: Probability of convergence to the optimal joint action in the general penalty game (FMQ/Q-learners).

to be considered by the FMQ-learner. Note that the move from 2-agent to 3-agent experiments increases the size of the joint action space by a factor of 3, from a total of 9 joint actions to 27.

One observation that is important is that, despite the limited success of the experiments with the FMQ-learner, the learning team still performed better than the same size team of only Q-learners. More importantly, the team with the added FMQ-learner performed better overall than the team without the FMQ-learner even though the two problems differ greatly in the size of the joint action space.

To illustrate this point, we have included a plot of the probability of convergence to the optimal joint action in teams with only Q-learners in the general penalty game. This is shown in Figure 5.

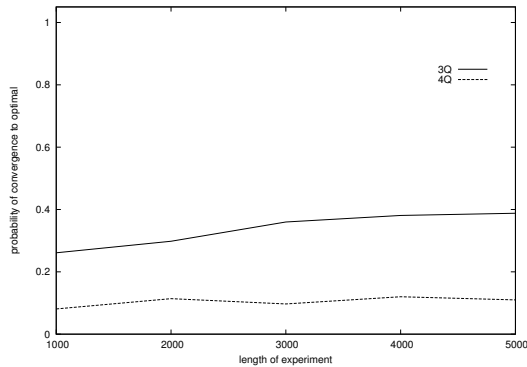


Figure 5: Probability of convergence to the optimal joint action in the general penalty game (Q-learners).

5.2 Evaluation with the Number-Matching Game

The general number-matching game was again less challenging than its penalty game counterpart. The addition of the FMQ-learner proved positive in all experiments, as is shown in Figure 6.

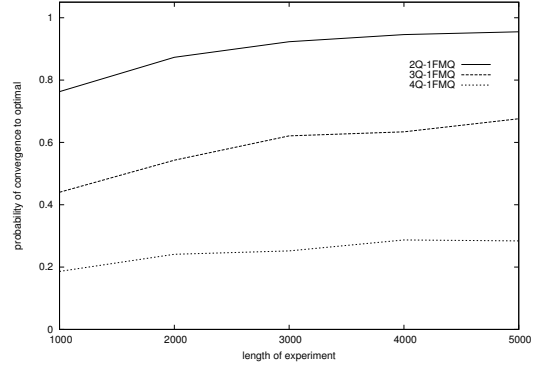


Figure 6: Probability of convergence to the optimal joint action in the general number-matching game (FMQ/Q-learners).

The same effect as before was observed in these experiments too. The addition of one FMQ-learner, although it causes the joint action space to grow significantly, it still provides better learning performance. We hope this is a general enough result to be exploited further in our future work.

5.3 Discussion

In the previous sections, we have investigated the effect that the addition of one or more FMQ-learning agents has to a group of Q-learners.

The rather surprising result is that adding an FMQ-learner is always beneficial. This is quite significant since the addition of another agent to the system makes the joint action space grow exponentially. When a 2-agent problem with 3 actions per agent had only 9 joint actions, a 3-agent problem with, again, 3 actions has 27 joint actions. The addition of an FMQ-learner to a group of agents means both that the agents converge more often to the optimal joint action and that they converge to better actions in general. This is a phenomenon that obtains in all games that we have tried.

However, another interesting phenomenon is that the agents need only polynomially more time to solve a problem that is exponentially bigger. Again, this is supported by our experimentation and will be evaluated further in our future work to show whether this observation obtains in general.

6 Limitations

Heterogeneity in agents is not merely a question of learning algorithm. Changing just one setting can turn a successful experiment into an unsuccessful one. For example, if we pair up a Q-learner and an FMQ-learner to solve the climbing game (see Table 1) and set the maximum temperature to 499, set the FMQ-learners confidence pa-

parameter to 5, set both learning rates to 0.9 and allow them to run for 1000 moves, they will converge to the optimal joint action (a, a) approximately 55% of the time. This may not be great but it is still better than if we allowed the Q-learner the use of a different temperature function. If the Q-learner was using a linear temperature function instead of the exponential one, the learners would not converge at all to the optimal joint action. The learners would, instead, consistently converge to a suboptimal action. This instability with respect to the degree of heterogeneity in the agents is an issue that has to be addressed in future research.

Finally, it is important to note that the addition of more than one FMQ-learner to the agent population does not improve results. In fact, it even tends to reduce performance a little for the same or slightly greater experiment length. The advantages of the addition of more "smart" agents seem to be outweighed by the exponential increase in the joint action space.

7 Conclusions and Outlook

We have presented an experimental study of the learning of coordination for heterogeneous multi-agent systems. Specifically, we have shown that a learning agent which employs the FMQ heuristic can achieve high levels of convergence towards an optimal joint action when teamed-up with one or more simple Q-learners. This has been shown on games where two or more simple Q-learners would not be able to achieve optimal coordination by themselves. In other words, the FMQ-learner "pushes" the simple Q-learners to the optimum. This result indicates that it is possible for a multi-agent system developer to achieve optimal coordination even when he/she does not have complete control over the nature of the agents that are going to be part of it.

While the results presented are very encouraging, there is still more work to be done in generalising our results. Specifically, we plan to study the performance of other heuristic learners when teamed-up with a wider range of other kinds of learning agents. Furthermore, we plan to investigate limitations, such as those mentioned in the previous section. We also intend to extend our studies to stochastic single-stage games (Kapetanakis et al., 2003), as well as multi-stage games.

References

- C. Boutilier. Sequential optimality and coordination in multiagent systems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 478–485, 1999.
- Georgios Chalkiadakis and Craig Boutilier. Coordination in multiagent reinforcement learning: A bayesian approach. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 709–716, Melbourne, Australia, 2003.
- Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 746–752, 1998.
- Drew Fudenberg and David K. Levine. *The Theory of Learning in Games*. MIT Press, Cambridge, MA, 1998.
- Leslie Pack Kaelbling, Michael Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 1996.
- Spiros Kapetanakis and Daniel Kudenko. Reinforcement learning of coordination in cooperative multi-agent systems. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 326–331, Edmonton, Alberta, Canada, 2002.
- Spiros Kapetanakis, Daniel Kudenko, and Malcolm Strens. Learning to coordinate using commitment sequences in cooperative multi-agent systems. In *Proceedings of the Third Symposium on Adaptive Agents and Multi-agent Systems (AAMAS03)*. Society for the study of Artificial Intelligence and Simulation of Behaviour, 2003.
- Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of the Seventeenth International Conference in Machine Learning*, 2000.
- Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In Morgan Kaufman, editor, *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163, San Mateo, CA, USA, 1994.
- Sandip Sen and Mahendra Sekaran. Individual learning of coordination knowledge. *JETAI*, 10(3):333–356, 1998.
- K. Verbeeck, A. Nowe, and K. Tuyls. Coordinated exploration in stochastic common interest games. In *Proceedings of Third Symposium on Adaptive Agents and Multi-agent Systems*, pages 97–102, University of Wales, Aberystwyth, 2003.
- Xiaofeng Wang and Tuomas Sandholm. Reinforcement learning to play an optimal nash equilibrium in team markov games. In *Proceedings of the 16th Neural Information Processing Systems: Natural and Synthetic conference*, Vancouver, Canada, 2002.

Evolving the Game of Life

Dimitar Kazakov*

*Department of Computer Science, University of York
Heslington, York YO10 5DD
kazakov@cs.york.ac.uk

Matthew Sweet†

†Department of Computer Science, University of York
Heslington, York YO10 5DD
mts104@cs.york.ac.uk

Abstract

It is difficult to define a set of rules for a cellular automaton (CA) such that creatures with life-like properties (stability and dynamic behaviour, reproduction and self-repair) can be grown from a large number of initial configurations. This work describes an evolutionary framework for the search of a CA with these properties. Instead of encoding them directly into the fitness function, we propose one, which maximises the variance of entropy across the CA grid. This fitness function promotes the existence of areas on the verge of chaos, where life is expected to thrive. The results are reported for the case of CA in which cells are in one of four possible states. We also describe a mechanism for fitness sharing that successfully speeds up the genetic search, both in terms of number of generations and CPU time.

1 Introduction

The aim of this work is to evolve a set (table) of rules for a cellular automaton (CA) with a good potential for producing “interesting” life forms, e.g., such that grow, move, have a long life span, consist of differentiated types of tissue, are compact and/or preserve their shape or produce a copy of themselves as a by-product of their growth. Rather than focussing on each such property in isolation, and trying to find the combination of rules that promotes it, an attempt is made to study an entropy-based criterion that is used as an indicator of the likelihood of such desirable life-forms appearing in a given CA. The criterion is used as a fitness function of a genetic algorithm (GA) searching through the range of possible CA. Also, to improve the GA performance on a task in which, intuitively, good solutions are few and far apart, we employ two techniques to maintain the balance between population quality and diversity — crowding (De Jong, 1975) and extended fitness — and show the substantial advantages that the latter brings in.

Cellular automata are dynamic systems consisting of a lattice of cells (in any number of dimensions) each of which has a number of associated discrete states k . The state of these cells is updated at discrete time steps, the resultant state dependent upon local state rules. Here we study two-dimensional automata with cells being in one of four possible states ($k = 4$, in other words, the cell belongs to one of three different types or is empty). The range of cells that influence the subsequent state of a cell is limited to immediate neighbours (sometimes denoted as $r = 1$ (Mitchell et al., 1993)). Each CA is defined by the table of rules that describe the subsequent state of the central cell for each 3×3 neighbourhood.

2 Genetic Algorithms and Extended Fitness

Genetic algorithms are search algorithms based on the mechanics of Darwinian evolution and genetics. GAs, despite their large variety, are all based on the same basic principles. They maintain a population of individuals representing candidate solutions to an optimisation problem. For each generation, a *fitness* reflecting estimated or actual quality is assigned to each solution (individual). A next generation of individuals is obtained by sampling the current so that individuals with higher fitness are favoured. Finally, the new generation is subjected to genetic operations such as crossover and mutation that aim at introducing a variety of new individuals. Then the cycle is repeated until some termination condition is fulfilled (Goldberg, 1989).

The fitness of an individual may measure the quality of the solution it proposes in absolute terms, say, as a scalar representing the value of a function that the GA is trying to maximise. In other cases, e.g., when it is normalised, fitness only represents the relative quality of an individual with respect to the rest. Fitness could be an even more abstract concept only reflecting the rank of the individual in the population.

Whether the goal of the GA is to provide a single best solution or a number of these, its principle remains the same: to store copies of the best one(s) aside from the main population, and wait until better ones are produced. For that ever to happen, it is essential that the GA should be capable of producing individuals that have not been seen in the previous generations. While the genetic operators, such as crossover and mutation, are the GA components that introduce new individuals in the population, their success depends on preserving sufficient genetic va-

riety in it. Evolution, whether in nature or as implemented in GAs, can be seen as a dynamic process driven by two factors : (natural) *selection*, which favours the survival of the fittest, and *genetic variation*, which introduces new individuals, some of which could potentially outperform the best so far. Neither factor is sufficient on its own : without selection, the search for the best individual will become completely random ; without genetic variation, there will be nothing that selection can act upon as a uniform population of identical individuals reaches a dead end.

GAs employ several techniques that study the individual's fitness in the context of the whole population and modify it to preserve the balance between the forces of selection and those increasing genetic variation. Fitness *scaling* is used to reduce the risk of clones of one 'superindividual' taking over the whole population in the early stages of the search when the individuals' fitness is very varied and generally low. Also, in a population with a minimum of variety in the fitness, scaling helps emphasise the existing differences and promote the best individuals more strongly. In either case, scaling aims at normalising the differences between the fitness of individuals with respect to the extremes present in the population. This could be done in a number of ways, from using a linear scaling function to ranking individuals according to their fitness and substituting rank for the original fitness.

Holland (1975) has observed that for fitness functions with a rugged landscape two high fitness parents often generate 'lethals' (very low fitness offspring). De Jong (1975) suggests that this effect can be combated using an algorithm with crowding factoring such that a new individual will replace an individual from the previous generation with a similar genetic make up. For each child, a subset of the population is selected at random that contains k individuals and the member of that set closest (by bitwise comparison) to the new offspring is replaced. In this model, k is known as the crowding factor, and was shown by De Jong to have an optimal value of 2 over the complex multimodal foxhole function. In this work, trials have shown that 10 is an optimal value for the crowding factor given an initial population size of 150.

Inbreeding with intermittent crossbreeding is a technique proposed by Hollstien (1971) for search using genetic algorithms with multimodal fitness functions. The idea is that the individuals in each niche mate until the average niche fitness ceases to rise and then the individuals in that niche can mate with individuals in different niche. This allows the neighbourhood of each local maximum to be thoroughly searched before guiding the search for a global maximum to another unexplored part of the search space.

Although not used here, another technique worth mentioning is *niching*, which is used to modify fitness in order to avoid overpopulating parts of the search space in favour of others. In general, this technique is based on partitioning the range of all individuals into *niches* and

```
procedure evaluateExtendedFitness
```

```
for each chromosome C do
| extFitness(C) = 0
|   for each locus L do
|   | extFitness(C,L) = 0
|   | numberOfMatches=0
|   |   for each chromosome C' do
|   |   | if locus L in C = locus L in C'
|   |   |   increment numberOfMatches
|   |   | _ extFitness(C,L) += fitness(C')
|   |   extFitness(C,L) /= (numberOfMatches *
|   |                           chromosomeLength )
|   _ _ extFitness(C) += extFitness(C,L)
```

Figure 1: Procedure computing extended fitness.

then reducing the fitness of individuals in overpopulated niches (Mahfoud, 1995).

Yet another alternative proposed here is the approach we call *extended fitness* (see Figure 1), in analogy to Dawkins's notion of extended genotype (Dawkins, 1982). As in population genetics, the components of this fitness are defined to measure the relative advantage that a gene gives to its carrier with respect to all other genes that can appear in the same locus. The fitness of the whole genome then can be computed as the averaged contribution of all its loci (Falconer, 1981). Extended fitness favours individuals with good genetic material (building blocks) and relies on the assumption that these could potentially be useful in the evolutionary search. Figure 2 shows the actual implementation of the way extended fitness is computed, which is faster, but more difficult to follow than the one shown in Figure 1. The overhead $O(\text{numberOfChromosomes} * \text{chromosomeLength})$ introduced by this latter implementation is very modest, and, as the experimental section of this article shows, it can be easily outweighed by the benefits it brings.

3 Evolving Cellular Automata

Using genetic algorithms for the search of CA with desired properties is a trend with a relatively short history. Previous research has often focussed on one-dimensional automata (Packard, 1988; Mitchell et al., 1993) or ones with two cell states ($k = 2$) (Packard, 1988; Mitchell et al., 1993; Sapin et al., 2003). In all cases, the resulting cellular automata (sets of rules) fall into one of four classes as defined by Wolfram (1983). Class 1 automata evolve after a finite number of time steps from almost all initial configurations to a single unique homogenous configuration in which all cells in the automaton have the same value. Class 2 automata generate simple disjoint structures dependent upon their initial configuration. The evolution of class 3 automata produces chaotic patterns

```

procedure evaluateExtendedFitness2

for each locus L do
for each allele A in L do
| fitness(L,A) = 0
| _ presence(L,A) = 0

for each chromosome C do
for each locus L in C do
for each allele A in L do
| fitness(L,A) += standardfitness(C)
| _ presence(L,A)++

for each locus L do
for each allele A do
| _ fitness(L,A) = fitness(L,A) / \
    ( presence(A) * chromosomeLength )

for each chromosome C do
| extFitness(C) = 0
| for each locus L in C do
| for the allele A in L do
| _ | _ extFitness(C) += fitness(L,A)

```

Figure 2: More efficient computation of extended fitness.

from almost all initial configurations; the statistical properties, after sufficient time steps, of the patterns produced by almost all initial configurations is the same. All other automata fall into a fourth class where for most initial configurations the automaton cells will become almost entirely unpopulated, however some stable live structures will develop which will persist indefinitely.

It would be unlikely that interesting creatures could exist in class 1 automata since after a finite time period all cells in the automaton would have the same value; therefore no interesting creatures could persist past this point. Class 2 automata merely generate simple disjoint structures (either periodic or static), which means that no reproducing or dynamic creatures could be created. Class 3 automata cannot support interesting creatures since the patterns produced are chaotic. Therefore the focus of this paper must be the fourth class of automata since they are most likely to be able to satisfy the criteria of supporting interesting creatures. The above speculations should be compared with Wolfram's suggestion that the fourth class of CA is of sufficient complexity to support universal computation Wolfram (1984).

When the fitness of a set of rules is to be determined then the cellular automaton that is represented by that set of rules needs to be run on some initial configuration of cells in the lattice. There are two techniques to consider that have been used in previous research to generate initial states — random generation and specific pattern generation. In random generation, some portion of the board is populated at random with live cells, and the density of live

cells is dependent upon the probability of each cell being live. In specific pattern generation, a user defined pattern is created usually at the centre of an otherwise unpopulated lattice.

In their work, Basanta *et al.* (Basanta, 2003) used a static initial state — only the central cell is live and all others are initially unoccupied. This reduces the amount of computation necessary for each fitness calculation. For an algorithm using randomised initial states, several runs are needed to attain an accurate fitness for the rule set, however with a single static initial state the fitness must only be calculated once.

The approach adopted here is based on two GAs. Each individual of the first GA encodes the rules of one CA. For each of these CA, another, nested, GA is used to search for an initial configuration of the given CA that has the highest possible fitness. The fitness of an initial configuration is computed by running the CA through a number of time steps, summing the fitness for each of them. This fitness is then used as the fitness of the CA. The inner genetic algorithm uses the fitness function described in the next section to discover initial configurations which favour the evolution of interesting life. All this can be summarised as follows:

1. Use a GA to select the best CA (set of rules).
2. To evaluate each CA, use another GA to select the best initial configuration (IC) and use its fitness as the CA fitness.
3. To evaluate each pair (CA,IC), run CA with IC for a predefined number of steps, measuring fitness at each step, summing it up, and returning the total. In other words, a CA, as defined by its set of rules, is only as fit as the fittest initial configuration that has been found for it.

The fitness landscape for this problem is highly rugged and therefore one must consider techniques for improving the effectiveness of the genetic algorithm under these conditions, with a particular attention to fitness scaling. In this work, crowding and extended fitness have been employed and compared.

4 Entropy Based Fitness of Cellular Automata

In this section, we introduce the fitness criterion used by the inner of the two above mentioned GAs.

The entropy of a system is defined to be the level of orderliness or chaos in that system — the higher the level of chaos, the higher the entropy. Wolfram (1983) defines the entropy of a CA to be:

$$S = - \sum_i p_i \log_2 p_i \quad (1)$$

where S is the entropy and p_i is the probability of state i . For two-dimensional automata an equation was developed by Wolfram and Packard (1985) to express the *set entropy* of an automaton.

$$S = \lim_{X,Y \rightarrow \infty} \frac{1}{XY} \log_k N(X, Y) \quad (2)$$

where S is the entropy, X and Y are dimensions of the area for which the entropy is being calculated and k is the number of different states. When used to calculate the entropy of a particular state, $N(X, Y)$ is the number of possible different states by which the current configuration can be represented. E.g., a 3×3 area containing one live cell could be represented by 9 different states, so $N(X, Y) = 9$. The division by XY normalises the entropy values, so that entropies over different tile sizes can be compared.

To calculate an approximation of this set entropy is far cheaper than to calculate the entropy using the first equation, since this first option would require us to enumerate all possible states. Also we are not interested in the overall automaton entropy but rather in the entropy of *tiles* (Sapin et al., 2003), a lattice of cells which is a component of the overall lattice of cells making up the automaton.

Another possible method of calculating the entropy would be to use the site entropy approach used by Langton. This is based on the entropy of a single cell of the lattice, and is defined to be 1 if the cell is in a different state to its state in the previous time step. The entropy of a tile therefore would be the sum of the entropies of all its constituent cells. To normalise the tile entropies, as with the set entropy calculation, it would be necessary to divide the tile entropy by the tile size, which gives the proportion of the cells that are in the same state as in the previous time state.

If we based the fitness function on the overall entropy then selecting for high entropy would result in a chaotic class 3 automaton which could not support interesting life. Conversely, selecting for a high degree of order (low entropy) would favour class 1 and class 2 automata which reach a stable state and never leave it. Interesting life is most likely to develop on the boundary between order and chaos; we need dynamic behaviour inherent in chaotic systems but we also need a degree of order to keep any developing life coherent. Therefore we wish to promote rule sets that contain both order and chaos — this can be achieved by assigning a *fitness based on the spread (standard deviation) of the entropies of (a sample of) the tiles making up the automaton*. We have also hoped that setting $k = 4$ would allow for specialisation among the types of cell, in a way specialised types of cells (tissue) have evolved in nature.

5 Results and Evaluation

A rule set was generated using a 100×100 board, and an initial population of 150 rule sets for the outer entropy

based GA, which was run for 300 generations. The inner genetic algorithm, evolving for a given rule set initial configurations that create interesting creatures, had a population of 10 and was run for 20 generations — the population size and number of generations have to be kept low since they have a large effect on the run time of the algorithm. To evaluate each initial configuration, the CA was run with the given set of rules and initial configuration for 200 steps. To compute the fitness of a configuration (CA board), only tiles of size 3×3 and 15×15 were considered.

The properties of the best rule set found have been empirically analysed and are as follows. No single cell of any colour can survive, and neither groups of cells which are of type 1 (red) or 2 (blue). Cells of type 3 (green) survive, unchanged and unproductive, in some compact formations of sufficient size. All other life seen so far under the rule set consists of more than one different type of cell (and this is observed, as a rule, whenever the experiment is repeated). We have produced a rule set which favours life consisting of different types of tissue, one of the aims of the work, without specifying this as part of the genetic algorithm.

Here are some examples of the ways different types of cell interact. Any cell of type 1 (red) requires another cell of the same type, as well as a cell of type 3 (blue) in its neighbourhood in order to survive. Red cells catalyse the growth of blue cells: any non-live cell adjacent to a red cell and a blue cell will grow into a live blue cell. As a result, a cluster of red and blue cells will gradually evolve into a connected core of red cells completely enclosed by the blue. Cells of type 3 (green) grow in the presence of other 4 of the same kind, as well as when a combination of 1 green and 1 red is present. Certain combinations of green and blue neighbours breed another blue cell, and often clusters of these two types of cell are stable or show periodic behaviour.

To summarise, one can see type 1 cells (red) as serving as an inert ‘skeleton’ which has to be protected by a layer of blue (type 2) cells; type 3 (green) cells promote periodic behaviour in combination with type 2, and can be grown themselves in the presence of red (type 1) cells. Figures 3–4 are examples of some of the small structures that survive in this CA. For configurations with sufficient density (e.g., 0.7) complex, connected structures such as the one in Figure 5 with stable and periodic components, spanning more than half the environment in each direction, emerge as a rule. Here size is an important factor. A large structure will show a periodic behaviour equal to the least common multiple of the periods of all dynamic substructures. For substructures of period 3, 4 and 5 (all observed), the overall period will be 60, etc.

Figure 6 shows the type of life produced by another rule set selected by our algorithm after 200 generations. This rule set started from almost any initial configuration will generate a pattern resembling that of animal spots or organs made of layers of tissue. The automaton is resilient

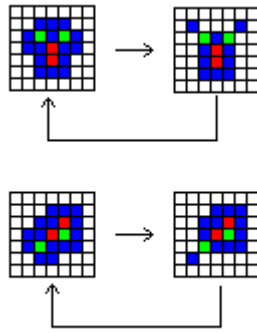


Figure 3: Cyclic structures with period 2.

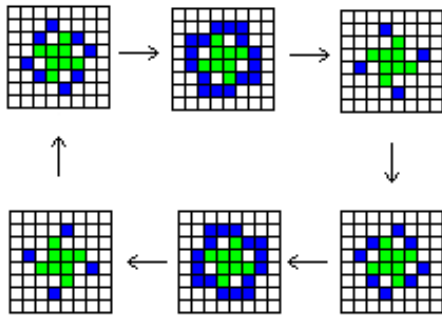


Figure 4: A cyclic structure with period 6.

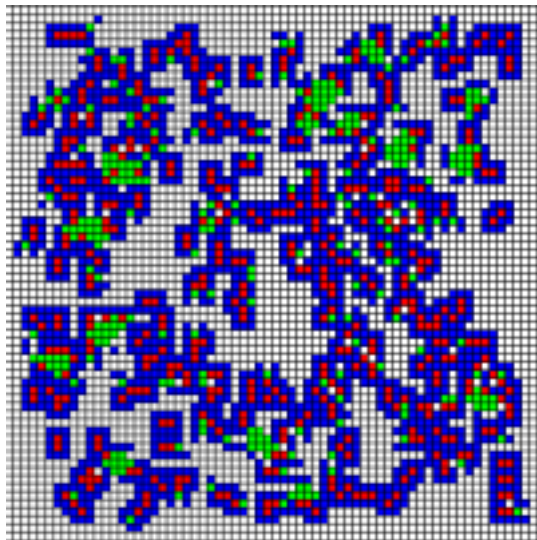


Figure 5: A large structure evolved from a dense, random configuration.

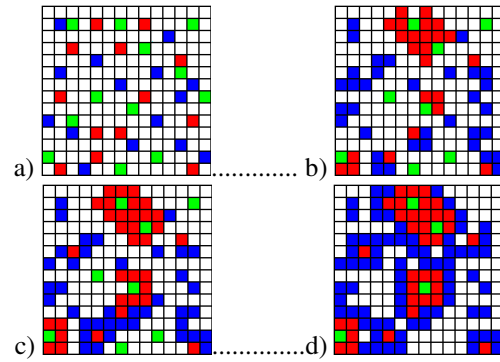


Figure 6: (a) Randomly generated initial configuration of the cellular automaton (density 0.2); (b) state after 1 transition; (c) state after 2 transitions; (d) state after a large number of transitions.

to damage and will regrow any killed cells so long as the core (green) cell is not removed.

Figures 7–8 show an interesting result related to the way GA is implemented: extended fitness produces faster improvements of the average population fitness despite the computational overhead it introduces. Moreover, extended fitness has a positive effect not only when the number of generations is compared, but also in terms of computational time, which is a very rewarding result. In a final observation on extended fitness (Figure 9), we have shown that the average population fitness can improve faster (again, in the stronger terms of time needed) with a larger population, which may seem counter-intuitive. This could be explained by the rugged fitness landscape which can be mapped more accurately by a larger population.

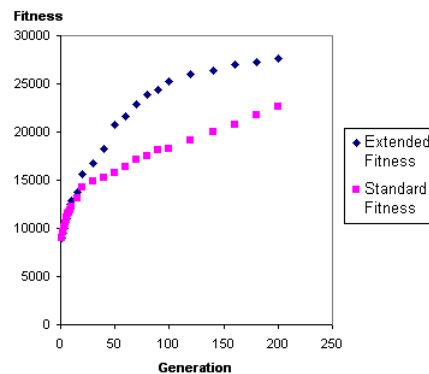


Figure 7: Comparison of the average fitness of rule sets for extended fitness and standard fitness approaches by generation (averaged over 10 runs using a population size of 100).

6 Identifying Moving, Reproducing Creatures in Cellular Automata

In order to identify rule sets that support interesting life forms, the population of the genetic algorithm with the entropy based fitness function at the final generation will be used as the initial population to a second genetic algorithm. This second algorithm evaluates CA with a fitness function that identifies individual creatures (connected groups of cells) and maintains a table of those which it has seen. Creatures are identified by the proportion of each type of cell of which they are made up. The creatures which have been seen are stored in a list, the members of which are compared with the board at regular intervals. If a creature is seen again, then its past and present positions are compared. If the creature has moved, then the CA fitness is increased by the number of cells making up that creature. As a result, the fitness promotes CA generating long-living creatures that move. Creatures can be composed a minimum of 5 and a maximum of 200 live cells to reduce computation costs.

The problem with this fitness function is that it is very expensive to compute taking nearly 600ms (on the 700 MHz test machine). If we were to substitute this fitness function for the entropy-based one, and run the experiment described at the beginning of Section 5, it would mean calling it around $8 \cdot 10^6$ times, which would take approximately 8 weeks. Instead, the genetic algorithm using this fitness function was used to further evolve a population of rule sets generated by the an initial genetic algorithm using the entropy based fitness function (see Figure 10).

The process for identifying creatures is as follows:

1. Initially create a new array the same size as the game board, called the creature search array, and initialise all cells to 'unchecked'.
2. Run through the game board sequentially checking each cell in the creature search array marked as 'unchecked'.
3. When a cell in the game board is discovered which contains life then check the surrounding unchecked cells to identify if any of those are live. As each cell is checked remember that, so that it is not looked at again.
4. Keep a count of each type of cell encountered in the creature stopping once no more live cells are found to be connected to the original live cell. Now check that the creature size is between the minimum and maximum size for an organism; if so, keep a record of it, in terms of the three different cell types.
5. Continue traversing the game board until each cell is recorded as 'checked'.

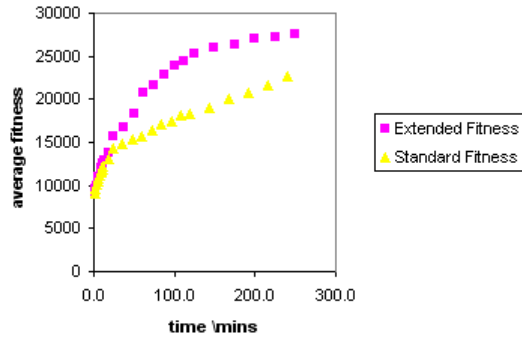


Figure 8: Comparison of the average fitness of rule sets for extended fitness and standard fitness approaches by time (averaged over 10 runs using a population size of 100).

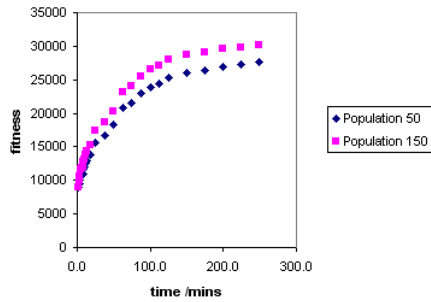


Figure 9: Comparison of the average fitness of rule sets for extended fitness and standard fitness approaches by population size (averaged over 10 runs).

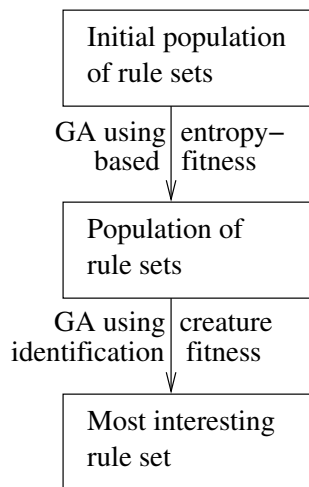


Figure 10: Generation of an interesting ruleset combining two different fitness measures.

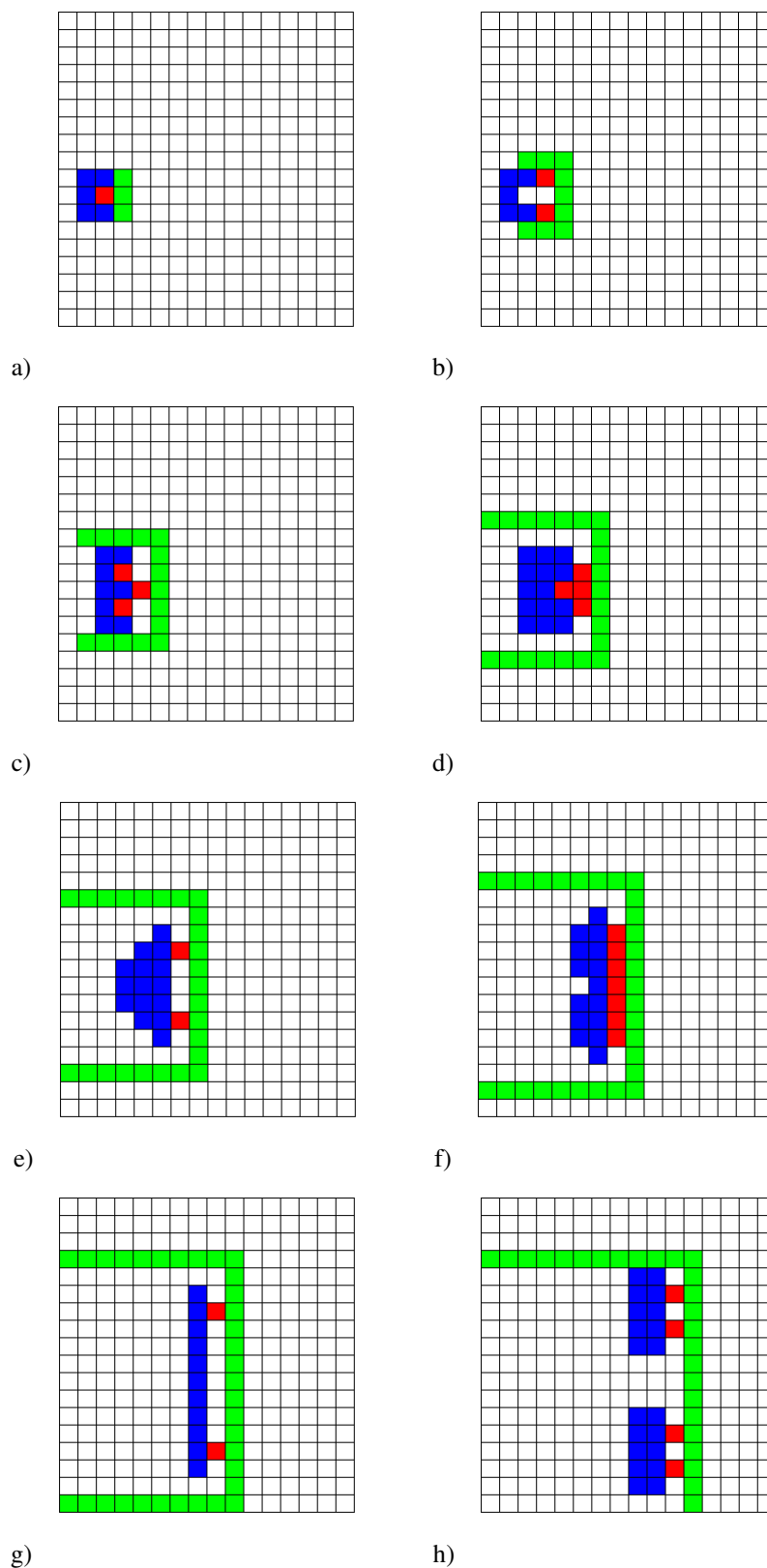


Figure 11: A trace of a creature over 8 consecutive steps. The creature moves to the right, expanding, and eventually splitting into two. Two very similar copies of the initial structure reappear in frame (g). In frame (h) they separate and the protruding 'nucleus' of each splits into two.

6. Now check to see if any of these creatures have been seen before since the start of this game: a creature with relative counts of each type of cell that fall within 5% of the counts of a previously seen creature, is seen as matching.

do this by dividing the number of cells containing each life type by the number of cells containing that type for the recorded creature. Then compare the three values produced which should also be within a set percentage (5% during testing) of each other.

7. If a creature has not been seen before then it must be added to the list of creatures.
8. Check the creature has moved from its previous position before incrementing the fitness by the number of live cells in that creature. If more than one copy of a single type of creature is seen in a turn then multiply the number of live cells in all organisms of that type by the number of the organism present. For instance, if there are three copies of an eight-celled creature then each creature has a fitness of 24 and so a total of 72 is added to the turn fitness for all three creatures.

Using this genetic algorithm to further evolve the rule sets from the final generation of the entropy based genetic algorithm several interesting creatures were discovered. One of the most interesting is a moving reproducing creature. The creature is resistant to damage and splits every 11 time steps. Another example of moving, reproducing creatures is shown in Figure 11.

7 Discussion and Further Work

The experiments described above show that the fitness function based on entropy did, indeed, generate non-trivial cellular automata. The addition of another fitness function promoting moving and reproducing creatures has also achieved its goal. Future work should continue to concentrate on automatically isolating and following the development of life-forms in a given CA. The behaviour of these creatures should be compared with the CA rules they are based on, in an attempt to find analogies with the processes of autocatalysis, mutual catalysis and inhibition, characteristic of biological systems.

Another achievement of this work is that it demonstrates the substantial benefits of using extended fitness on a task with an apparently very complex fitness landscape.

For the chosen type of CA (two-dimensional, $k = 4$ and $r = 1$), there are hundreds of rules in each rule set. Such large number of rules means it is difficult to analyse all the pathways in which cells can interact. One way to deal with this issue is to use a machine learning technique to summarise all rules and represent them in a more expressive formalism, stating, for instance, “cell of type X

will be created if 2 to 4 cells of type Y are present”, rather than enumerating these cases separately. Inductive Logic Programming (ILP) is an excellent candidate for the task Muggleton and Raedt (1994).

References

- D. Basanta. Evolving automata to grow patterns. *Symposium on Evolvability and Interaction*, 2003.
- R. Dawkins. *The Extended Phenotype*. Oxford University Press, 1982.
- K.A. De Jong. *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.
- D.S. Falconer. *Introduction to Quantitative Genetics*. Longman, London, second edition, 1981.
- D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
- J.H. Holland. *Adaption in natural and artificial systems*. University of Michigan Press, 1975.
- R.B. Hollstien. *Artificial Genetic Adaption in Computer Control Systems*. PhD thesis, University of Michigan, 1971.
- S. Mahfoud. *Niching Methods for Genetic Algorithms*. PhD thesis, University of Illinois, Urbana-Champaign, 1995.
- Melanie Mitchell, Peter T. Hraber, and James P. Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*, 7:89–130, 1993.
- S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.
- N. H. Packard. *Dynamic Patterns in Complex Systems*, chapter Adaptation towards the Edge of Chaos. World Scientific, 1988.
- E. Sapin, O. Bailleux, and J. Chabrier. Research of complex forms in the cellular automata by evolutionary algorithms. In P. Liardet et al., editor, *Proc. of the 6th Intl. Conf. on Artificial Evolution*, Marseille, Oct 2003.
- S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55, 1983.
- S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.
- S. Wolfram and N. Packard. Two-dimensional cellular automata. *Statistical Physics*, 38:901–946, 1985.

Dynamic and Distributed Interaction Protocols

Jarred McGinnis* and David Robertson*

*Centre for Intelligent Systems and their Applications
University of Edinburgh
Appleton Tower, Room 4.15
Edinburgh EH8 9LE
j.p.mcginis@sms.ed.ac.uk

Abstract

This paper describes a protocol language which can provide agents with a flexible mechanism for coherent dialogues. The protocol language does not rely on centralised control or bias toward a particular model of agent communication. Agents can adapt the protocol and distribute it to dialogical partners during interactions.

1 Introduction

As the programming paradigm of agency evolves, more robust, diverse, and complex agents are developed. The growing heterogeneity of agent societies will increase even further as the research and development of deliberative and communicative models produce new and interesting approaches (Pasquier et al., 2003). The need for an equally adaptive means of communication between this heterogeneous multitude also grows.

Electronic Institutions (Estava et al., 2001) and other state-based approaches are not feasible for use in open multi-agent systems with dynamic or large conversation spaces. The term conversation space is used to express every possible sequence and combination of messages that can be passed between two or more agents participating in a given agent system. Protocols provide a useful framework for agent conversations and the concern that they sacrifice agent autonomy is exaggerated. In social interactions, humans and agents must willingly sacrifice autonomy to gain utility. If I want my train tickets or cup of coffee, I must follow the implicit protocol and join the queue. It is the same for software agents. If the agent must gain a resource only available by participating in an English auction, it behoves the agent to adopt the protocol necessary for participation in the auction. Whether this is done by an explicitly defined protocol or the agent learning the protocol implicitly makes no difference to the agent's behaviour within the system.

Electronic Institutions take a societal approach to agent communication. Control is top-down. Administrative agents perch above the system and keep an eye on the agents as they interact inside the system. Agent-centric approaches build systems bottom-up. These approaches attempt to pack individual agents with a model of communication which can react to a multi-agent system. They have a communicative model that sits besides or is intertwined with its rational model. This is done in a number of ways. The most common is a BDI-model of commu-

nication as typified by the standardisation organisation, FIPA. There is a lot of dissatisfaction with FIPA ACL, and a variety of alternative models for agent communication have been proposed. All trying to address faults perceived with the FIPA approach.

The protocol language described in this paper seeks a balanced approach. It utilises the useful aspects of Electronic Institutions without relying on administrative agents or statically defined protocol specifications. Agents communicate not only individual messages but the protocol and dialogue state as well. The use of protocols provides structure and reliability to agent dialogues. Yet, by describing protocols as a process rather than a fixed state-based model, the conversation space can be defined as the agent interaction progress rather than being statically defined during the engineering process. Distributing the protocol along with the message also allows agent to communicate the convention for communication as well as coordinate the dialogue.

Section 2 will discuss some of the dominant agent communication paradigms. Section 3 describes the syntax and features of the protocol language. A discussion of adaptable protocols in section 4 is followed by an example illustrating an adaptable protocol using dialogue games in section 5. Section 6 concludes the paper enumerating the accomplishments and potential issues associated with the approach.

2 Approaches to Agent Communication

2.1 Electronic Institutions

Electronic Institutions(EI) provide structure to large and open multi-agent systems(MAS). By emulating human organizations, Electronic Institutions provide a framework which can increase interoperability. The EI framework formally defines several aspects of an agent soci-

ety. The core of an EI is the formal definition of roles for agents, a shared dialogical framework, the division of the Institution into a number of scenes and a performative structure which dictates, via a set of normative rules, the relationships between the scenes. Agents interact with an Institution through the exchange of illocutions, i.e. messages with intentional force.

Participating agents are required to adopt a role within the Institution. This is similar to our entering a shop and assuming the role of a customer, and the employee adopting the role of salesperson. A role is defined as a finite set of dialogical actions. By the adoption of a role within an Institution, an agent's activities within the Institution can be anticipated. This abstraction of agents as a role allows the Institution to regulate and identify agent activities without analysing individual agents. Relationships between agents can be dealt with as generalizations. A role can be defined as subsuming or being mutually exclusive to another role.

The dialogical framework provides a standard for communication. Agents are guaranteed to have a shared vocabulary for communication as well as a common world-view with which to represent the world they are discussing. The dialogical framework is defined as a tuple consisting of an ontology, a representation language, a set of illocutions, and a communication language. The representation is an encoding of the knowledge represented by the ontology and makes up the inner language. This is contained within an individual illocution that is passed between agents. The illocution, as part of the outer language or communication language, expresses the intention of the agent by its communicating the message of the inner language. The dialogical framework, which contains the ontological elements, is necessary for the specification of scenes.

All interactions between agents occur within the context of scenes. Scenes are interaction protocols between agent roles. They are expressed as a well-defined protocol which maps out the conversation space between two agent roles. These scenes are represented as graphs. The nodes are conversation states and arcs representing the utterances of illocutions between the participants. Each scene will have a set of entrance and exit states with conditions that must be satisfied before the agent can begin or exit a scene. A set of roles and scene states are formally defined. An element of the set of states will be the initial state and a non-empty subset will be final states. Between the states there is a set of directed and labelled edges.

Scenes are individual agent conversations. In order for agents to participate in more interesting activities, it is necessary to formalize relationships between these individual conversations. The performative structure formalizes this network of scenes and their association with each other. The roles an agent adopts and the actions of the agents create obligations and restrictions upon the agent. These obligations restrict the further movement of agents. The performative structure is made of a finite non-empty

set of scenes. There is a finite and non-empty set of transitions between these scenes. There is a root scene and an output scene. Arcs connect the scenes of the Institution. These arcs have different constraints placed upon them. For example, the constraints can synchronize the participating agents before the arc can be fully traversed, or there are constraints that provide an agent a choice point upon which scene to enter.

Within the scenes of an Electronic Institution, the actions an agent performs affect the future actions available to the agent. These consequences can extend beyond the current scene. These consequences could be the requirement for an agent to perform an action in some future scene or even which scenes or sequence of scenes an agent is now required to be a participant. These normative rules are categorized between two types. *Intra-scene* dictate actions for each agent role within a scene, and *inter-scene* are concerned with the commitments which extend beyond a particular scene and into the performative structure (Esteva et al., 2000).

Tools (Esteva et al., 2002) exist to aid in the creation of the various components and development of Electronic Institutions. This includes a tool to verify any specifications developed as well as tools to aid the synthesis of agents that can participate in the Electronic Institution (Vasconcelos, 2002).

2.2 Agent-centric Design

FIPA ACL for better or for worse has made a large impact of agent communication research. A victim of its own success, most new approaches to agent communication are attempts to redress FIPA's ACL deficiencies. Conversation Policies (Greaves et al., 2000) were an attempt to produce a more 'fine-grained' means of generating dialogues. More recently, researchers have developed communicative models to address the semantic verification problem of FIPA ACL (Wooldridge, 2000). There are other approaches which see the importance of separating the agent's internal states from the conversational model (Maudet and Chaib-draa, 2002). Two approaches of interest are theories based on social commitment or obligation and formal definitions of agent systems based on dialogue theory.

2.2.1 FIPA ACL

The Foundation for Intelligent Physical Agents develops software standards for agent communication. This is expressed in their official mission statement: *The promotion of technologies and interoperability specifications that facilitate the end-to-end interworking of intelligent agent systems in modern commercial and industrial settings*. In practice, this includes the publishing of standards concerning speech acts, predicate logic, and public ontologies. The individual communicative acts of FIPA's Agent Communication Language (ACL) is based on the speech

act theory of Searle (1969). The semantics of FIPA ACL are based on the Belief-Desire-Intention (BDI) model of agency and is formalised in a language SL (for Intelligent Physical Agents, 2000).

Each communicative action by a FIPA compliant agent implies that agent is following the requirements specified for that action. This includes general properties for all communicative acts, the interaction protocol of which the act is a part, and the feasible preconditions and rational effects for that particular act (FIPA, 2001). For example, an agent i must believe a proposition p and believe that an agent j neither has any amount of belief about p or not p before it can send an **inform** FIPA ACL communicative act to agent j . Afterwards, agent i is entitled to believe that agent j believes p .

2.2.2 Social Commitment

Researchers have adopted the idea of social commitments to redress the semantic difficulties that arise when agents rely on mentalistic (e.g BDI) Agent Communication Languages (ACL). Social-based semantics consider the agent's relationship to its communicative partners. It is a recognition that an agent's communicative acts do not exist in a vacuum. It is the use of the intuitive idea that an agent's communication is an event which necessarily involves other agents.

Singh (2000) identifies several criteria for the semantics of an Agent Communication Language. According to Singh, an ACL should be formal, declarative, verifiable, and meaningful. To this end, he has developed a *social semantics*. He defines three facets to every communicative act. The *objective claim* which commits an agent to another that some proposition p holds. The *subjective claim* is that an agent believes p , and the *practical claim* that the agent has some justification or reason for believing p . This is a novel approach, because most reactions to the semantic verification problem of the mentalistic approach is to completely throw it away. Singh has, instead, embraced the mentalistic approach but coupled it with the idea of social commitment. The purely mentalistic approach rests on the assumption that the agent is sincere about p , but Singh has added that the agent is also socially committed to being sincere about p . It is recognized that the use of social semantics does not replace the need for protocols, but the combination of social semantics and protocols would create a much more flexible ACL Maudet and Chaib-draa (2002).

The approach described in Flores and Kremer (2002) uses the commitment themselves to develop the conversation between two agents. Flores argues that our verbal utterances carry with them obligations dependent on the role of the agent within a society. The question 'What time is it?' carries with it the obligation (in polite society) to not only reply but make an attempt to actually find out the time. The use of social commitments in multi-agent communication is to provide a number of rules that dic-

tate appropriate illocutions and actions performed based on the agent voluntarily obligating itself to commitments with other agents and eventually discharging those commitments. A protocol is defined for the negotiation of the adoption of social commitments. Agents propose to add and remove commitments for action from personal commitment stores. An agent will propose to add a commitment to perform some actions. Once this is accepted and the commitment is satisfied the protocol includes steps to propose the release of any further commitment to that action. It is through this simple protocol and the social commitment-based conversation policies an agent conversation can be developed.

2.2.3 Dialogue Theory and Games

The philosophers Doug Walton and Erik Krabbe have developed a typology of dialogues to detect fallacious reasoning (Walton and Krabbe, 1995). This typology was adopted by Chris Reed (Reed, 1998) in a formalism for multi-agent systems and inter-agent communication. Of the six kinds of dialogue identified, five of these dialogue types are applicable to the domain of agent communication. The sixth, *eristic*, is a dialogue where reasoning has ceased and the participants use the dialogue for the airing of grievances and one-upmanship. This dialogue type is important for the study of human conversations, but it is ignored by the agent research community. Dialogues are classified into the different types by three criteria. The first criterion considers the initial situation. What information does each of the participants have? Are the agents cooperative or competitive with each other? The second criterion concerns the individual goals an agent has for the interaction, and the third criterion are the goals shared by the participating agents. In *Information-Seeking* dialogues, one agent seeks the answer to a question which it believes the other agent possesses. *Inquiry* dialogues occur when two agents work together to find the answer to a question whose solution eludes both agents. A *Persuasion* dialogue has one agent attempting to convince another to adopt some proposition which it currently does not believe. *Negotiation* dialogues occur when the participants haggle over the division of a scarce resource. In *Deliberation* dialogues, the agents attempt to agree on a course of action for a particular situation. It is rare that any actual dialogue will be purely of one instance of one kind of dialogue. It is more likely that a dialogue will consist of an amalgamation of the different types. For example, during a negotiation, propositions may need clarification and an information-seeking dialogue would occur. This dialogue typology is fundamental to recent agent communicative models using dialogue games.

Dialogue games have existed for thousands of years, since Aristotle, as a tool for philosophers to formalise argumentation. It is an attempt to identify when an argument or its justification is weakened or undercut by an argument or refutation made by the other participant. By

each player making ‘moves’ and following a set of rules, it was hoped that properties of good and bad arguments could be identified. This formalism for argumentation has been employed to increase the complexity and robustness of software agents conversations. The objective is to produce a meaningful interaction between dialogical partners by following the rules of an individual dialogue game.

There are several components to a dialogue game. Firstly, the participants must share a set of locutions. This is a common requirement for models of agent communication. The commencement and termination rules specify the conditions under which a dialogue can start or end. This is a set of performatives from an agent communication language that is shared between the agents. This language must include the ability to utter assertions as well as justifications and challenges to those assertions. Another component is the combination rules. These rules define when particular illocutions are permitted, required, or illegal. The last part necessary for a dialogue game is the rules for commitment. These rules create obligations on the agent with respect to the dialogical moves of the agent. These commitments can be divided into dialogical and semantic. Dialogical commitments are the obligation of an agent to make a particular move within the context of the dialogue game. Semantic commitments indenture the agent to an action beyond the dialogue game itself. A record of these commitments is publicly stored. For example, if you say you are willing to pay the highest price in an auction, it will be known that you are committed to actually pay that price.

Dialogue game frameworks (McBurney and Parsons, 2002; Maudet and Evrard, 1998) attempt to construct more complex and robust agent conversations. This is achieved by combining different atomic dialogue types which have been identified by philosophers analysing human dialogues (Walton and Krabbe, 1995). This approach avoids the semantic ambiguities inherent in mentalistic models and the rigidity of static protocol-based approaches (FIPA, 2001). The dialogue game approach depends on several assumptions about participating agents. Agents participating in the dialogue game framework must agree on all the rules of the framework. The number of requirements made on individual agents in order for them to play dialogue games makes the approach unsuited for open multi-agent systems.

3 The Protocol Language

The development of the protocol language is a reaction Electronic Institutions (Walton and Robertson, 2002). Although the EI framework provides structure and stability to an agent system, it comes at a cost. Integral to EI is the notion of the administrative agents. Their task is to enforce the conventions of the Institution and shepherd the participating agents. Messages sent by agents are sent through the EI. This synchronises the conversation be-

tween the conversing agents, and keeps the administrative agent informed of the state of the interaction

An unreliable keystone makes the whole of the arch defective, just as the system is now dependent on the reliability and robustness of its administrative agent. Also, this centralisation of control runs counter to the agent paradigm of distributed processing. Within the scenes of Electronic Institutions, interaction protocols are defined to guarantee that agents utter the proper illocutions and utter them at the appropriate time. This is defined formally by the specifications of the EI and left to the designers of individual agents to implement. It assumes that the agent’s interaction protocol covers the entire conversation space before the conversation occurs. If the interaction needs of the institution change, this would require redefinition of the Institution and re-synthesis of the individual agents. Agents are also expected to know the global state of the system and their exact position within it. In EIs this is handled by an administrative agent whose job it is to synchronise the multitude of agents involved.

The protocol language addresses some of these shortcomings of EIs but retains the benefits of implementing the EI framework. Its goal is to lessen the reliance on centralised agents for synchronisation of individual participants in the system, provide a means for dissemination of the interaction protocol and the separate the interaction protocol from the agent’s rationalisations to allow the dynamic construction of protocols during the interaction. By defining interaction protocols during run-time, agents are able to interact in systems where it is impossible or impractical to define the protocol beforehand. The protocol language defined in Figure 1 is similar to the protocol language described in Walton (2004b) for which the formal semantics have been defined.

P	\in	Protocol	$::$	$\langle S, A^{\{n\}}, K \rangle$
A	\in	Agent Clause	$::$	$\theta :: op.$
θ	\in	Agent Definition	$::$	agent (r, id)
op	\in	Operation	$::$	null
		(Precedence)		θ
		(Send)		(op)
		(Receive)		$M \Rightarrow \theta$
		(Sequence)		$M \Leftarrow \theta$
		(Choice)		$op1 \text{ then } op2$
		(Parallelism)		$op1 \text{ or } op2$
		(Consequence)		$op1 \text{ par } op2$
		(Prerequisite)		$\psi \leftarrow M \Leftarrow \theta$
M	\in	message	$::$	$M \Rightarrow \theta \leftarrow \psi$
ψ	\in	state	$::$	$\langle m, P \rangle$
				a predicate

Figure 1: The abstract syntax of the protocol

Figure 1 defines the syntax of the protocol language. An agent protocol is composed of an agent definition and an operation. The agent definition individuates the agents participating in the conversation (id), and the role the agent is playing (r). Operations can be classified in

three ways: actions, control flow, and conditionals. Actions are the sending or receiving of messages, a no op, or the adoption of a role. Control Flow operations temporally order the individual actions. Actions can be put in sequence (one action must occur before the other), in parallel (both action must occur before any further action), or given a choice point (one and only one action should occur before any further action). Conditionals are the preconditions and postconditions for operations. The message passed between two agents using the protocol consists of three parts. The first is the actual illocution (m) the agent is wishing to express. The second is the full protocol (P) itself. This is the protocol for all agents and roles involved in the conversation. This will be necessary for the dissemination of the protocol as new agents enter the system. Other aspects of the protocol are the inclusion of constraints on the dialogue and the use of roles. An agent's activities within a multi-agent system are not determined solely by the agent, rather it is the relationship to other agents and the system itself that helps determine what message an agent will send. These can be codified as roles. This helps govern the activity of groups of agents rather than each agent individually. Constraints are marked by a ' \leftarrow '. These are requirements or consequences for an agent on the occurrence of messages or the adoption of roles. The constraints provide the agent with a shared semantics for the dialogue. These constraints communicate meaning and implication of the action to the agent's communicating partner. For example, an agent receiving a protocol with the constraint to believe a proposition s upon being informed of s can infer that the agent sending the protocol has a particular semantic interpretation of the act of informing other agents of propositions. The ' \leftarrow ' and ' \Rightarrow ' mark messages being sent and received. On the left-hand side of the double arrow is the message and on the right-hand side is the other agent involved in the interaction.

An agent must be able to understand the protocol, the dialogue state, and its role within the protocol. Agents need to be able to identify the agent clause which pertains to its function within the protocol and establish what actions it must take to continue the dialogue or what roles to adopt.

3.1 Implementing the Protocol Framework

A message is defined as the tuple, $\langle m, P \rangle$. Where m is the message an agent is currently communicating, and P is the remainder is the protocol written using the language described in figure 1. The protocol, in turn, is a triple, $\langle S, A^{\{n\}}, K \rangle$. S is the dialogue state. This is a record of the path of the dialogue through the conversation space and the current state of the dialogue for the agents. The second part is a set of agent clauses, $A^{\{n\}}$, necessary for the dialogue. The protocol also includes a set of axioms, K , consisting of common knowledge to be publicly known between the participants. The sending of

the protocol with the messages allows agents to represent the various aspects of Electronic Institutions described in section 2.1. In addition, agents themselves communicate the conventions of the dialogue. This is accomplished by the participating agents satisfying two simple engineering requirements. Agents are required to share a dialogical framework. The same is required of Electronic Institutions, and is an unavoidable necessity in any meaningful agent communication. This includes the requirements on the individual messages are expressed in a ontology understood by the agents. The issue of ontology mapping is still open, and its discussion extends beyond the scope of this paper. The second requirement obligates the agent to provide a means to interpret the received message and its protocol. The agent must be able to unpack a received protocol, find the appropriate actions it may take, and update the dialogue state to reflect any actions it chooses to preform.

Figure 2 describes rule for expanding the received protocols. Details can be found in Robertson (a). A similar language for web services is described in Robertson (b). An agent receives a message of the form specified in figure 1. The message is added to the set of messages, M_i , currently being considered by the agent. The agent takes the clause, C_i , from the set of agent clauses received as part of P . This clause provides the agent with its role in the dialogue. The agent then expands C_i by the application of the rules in figure 2. The expansion is done with respect to the different operators encountered in the protocol and the response to M_i . The result is a new dialogue state, C_n ; a set of output messages, O_n and a subset of M_i , which is the remaining messages to be considered, M_n . The result is arrived at by applying the rewrite rules. The sequence would be similar to figure 3. C_n is then sent as part of P which will accompany the sending of each message in O_n .

3.2 Features of the Protocol

Several features of the protocol language are useful for agents capable of learning and adapting to the multi-agent system in which they participate. Sending the dialogue state during the interaction provides agents with several advantages. It is no longer necessary for an administrative agent to shepherd the interaction. The sending of the protocol with the message uses the 'hot potato' approach to communication. The interaction is coordinated by which agent currently 'holds' the protocol. The reception of a message would cue an agent to action. The sending of the protocol provides a means for dissemination of the social conventions for the dialogue. The most common approach is to use specifications to be interpreted by individual engineers. The protocol directly communicate the social conventions and expectations an agent has for the dialogue. Agents with the ability to learn could use the received protocol to plan ahead or modify its own social conventions to be able to communicate with other agents.

$$\begin{aligned}
& A :: B \xrightarrow{M_i, M_o, \mathcal{P}, O} A :: E \\
& \text{if } B \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
& A_1 \text{ or } A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
& \text{if } \neg \text{closed}(A_2) \wedge A_1 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
& A_1 \text{ or } A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
& \text{if } \neg \text{closed}(A_1) \wedge A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
& A_1 \text{ then } A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \text{ then } A_2 \\
& \text{if } A_1 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
& A_1 \text{ then } A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} A_1 \text{ then } E \\
& \text{if } \text{closed}(A_1) \wedge A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
& A_1 \text{ par } A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O_1 \cup O_2} E_1 \text{ par } E_2 \\
& \text{if } A_1 \xrightarrow{M_i, M_n, \mathcal{P}, O_1} E_1 \wedge A_2 \xrightarrow{M_n, M_o, \mathcal{P}, O_2} E_2 \\
& C \leftarrow M \leftarrow A \xrightarrow{M_i, M_i - \{M \leftarrow A\}, \mathcal{P}, \emptyset} c(M \leftarrow A) \\
& \text{if } (M \leftarrow A) \in M_i \wedge \text{satisfy}(C) \\
& M \Rightarrow A \leftarrow C \xrightarrow{M_i, M_o, \mathcal{P}, \{M \Rightarrow A\}} c(M \Rightarrow A) \\
& \text{if } \text{satisfied}(C) \\
& \text{null} \leftarrow C \xrightarrow{M_i, M_o, \mathcal{P}, \emptyset} c(\text{null}) \\
& \text{if } \text{satisfied}(C) \\
& \text{agent}(r, id) \leftarrow C \xrightarrow{M_i, M_o, \mathcal{P}, \emptyset} a(R, I) :: B \\
& \text{if } \text{clause}(\mathcal{P}, a(R, I) :: B) \wedge \text{satisfied}(C)
\end{aligned}$$

A protocol term is decided to be closed, meaning that it has been covered by the preceding interaction, as follows:

$$\begin{aligned}
& \text{closed}(c(X)) \\
& \text{closed}(A \text{ or } B) \leftarrow \text{closed}(A) \vee \text{closed}(B) \\
& \text{closed}(A \text{ then } B) \leftarrow \text{closed}(A) \wedge \text{closed}(B) \\
& \text{closed}(A \text{ par } B) \leftarrow \text{closed}(A) \wedge \text{closed}(B) \\
& \text{closed}(X :: D) \leftarrow \text{closed}(D)
\end{aligned}$$

$\text{satisfied}(C)$ is true if C can be solved from the agent's current state of knowledge.

$\text{satisfy}(C)$ is true if the agent's state of knowledge can be made such that C is satisfied.

$\text{clause}(\mathcal{P}, X)$ is true if clause X appears in the dialogue framework of protocol \mathcal{P} , as defined in Figure 1.

Figure 2: Rules for expanding an agent clause

$$\langle C_i \xrightarrow{M_i, M_{i+1}, \mathcal{P}, O_i} C_{i+1}, \dots, C_{n-1} \xrightarrow{M_{n-1}, M_n, \mathcal{P}, O_n} C_n \rangle$$

Figure 3: Sequence of rewrites

The protocol language is strictly concerned with the interaction level of communication. The semantics of the language does not depend on any assumptions about the agent's internal deliberative model. All requirements for the interaction are publicly specified with the protocol. Agents with different models of deliberation are able to

communicate (McGinnis et al., 2003).

4 Means of Adaptation

Protocols are traditionally seen as a rigid ordering of messages and processing to enable a reliable means of communication. Agent-centric approaches have tended to avoid their use, lest agents be reduced to nothing more than remote function calls for the multi-agent system. The control over agent interactions within an electronic institutions is indeed intrusive. As described in section 2.1, the administrative agents of electronic institutions have complete control. The sequence of messages are dictated but also the roles an agent may adopt and the actions an agent must take within and outside of the context of the dialogue.

The protocol language of this paper does not follow this tradition. It is designed to bridge the gap separating the two approaches to agent interaction. The language is capable of representing the scenes and performative structure of electronic institutions, but it is not limited to electronic institution's inflexible model of agent interaction. The protocol language and the process of sending the protocol during execution provides agents with a means of adaptation.

In the electronic institution model, the protocol does not exist within the participating agents. It is retained by the institution itself, and designers must engineer agents that will strictly conform to the protocol which will be dictated by the administrative agents. Our approach delivers the protocol to the participating agents. Individual agents are given providence over the protocol they receive. This returns the power of the interaction to the participating agents. For example, the protocol received is not required to be the protocol that is returned.

The protocol, as described so far, already allows for a spectrum of adaptability. At one extreme, the protocol can be fully constrained. Protocols at this end of the spectrum would be close to the traditional protocols and electronic institutions. By rigidly defining each step of the protocol, agents could be confined to little more than remote processing. This sacrifice allows the construction of reliable and verifiable agent systems. At the other extreme, the protocols would be nothing more than the ordering of messages or even just the statement of legal messages (without any ordering) to be sent and received. Protocols designed this way would be more akin to the way agent-centric designers envisage agent communication. Agents using these protocols would be required to reason about the interaction to determine the next appropriate step in the dialogue. Though the protocol language is expressive enough for both extremes of the spectrum, the bulk of interactions are going to be somewhere in the middle. A certain amount of the dialogue will need to be constrained to ensure a useful dialogue can occur. This allows agents to express dynamic and interesting dialogues.

The protocol language is flexible enough to be adapted during run-time. Yet, protocols modified indiscriminately would return us to the problem facing the agent-centric approach. We would have a model for flexible communication, but no structure or conventions to ensure a meaningful dialogue can take place. It is necessary to constrain any adaptation in a meaningful way. By the examination of patterns and standards of an agent-centric approach, protocols can be constructed to have points of flexibility. Portions in the dialogue can be adapted without losing the benefits of a protocol-based approach. The example below employs the rules for playing a dialogue game, the protocol language, and an amendment to the rewrite rules to allow a more dynamically constructed protocol.

5 Example

Figure 4 shows an example of an Information-seeking dialogue game similar to the one defined in Parsons et al. (2003). The dialogue game rules are simplified to clarify its implementation within the protocol. There are countless variations on the rules for any one type of dialogue game. This illustrates a continuing problem with agent-centric communication design. It is not a trivial requirement to ensure agents within a system are employing the same communicative model. This is the same with dialogue games. Subtle differences could break the dialogue. By the use of the protocol, agent can communicate their ‘house’ rules for the game. The rules for this particular game are as follows:

1. The game begins with one agent sending the message *question(p)* to another agent.
2. Upon receiving a *question(p)* message, an agent should evaluate *p* and if it is found to be true, the agent should reply with *assert(p)* else send an *assert(null)* which is a failure message.
3. Upon receiving an *assert(p)*, an agent should evaluate the assertion, then the agent can send an *accept(p)* or *challenge(p)* depending on whether the agent’s acceptance attitude will allow.
4. Upon receiving a *challenge(p)*, an agent should send an *assert(S)*. *S* is a set of propositions in support of *p*.
5. For each proposition in *S*, repeat steps 3 and 4.
6. The game is over when all propositions have been accepted or no further support for a proposition can be offered.

Rule one is satisfied by an agent taking up the role of the ‘seeker’. This provides the agent with the legal moves necessary to play that side of the information-seeking dialogue game. The other agent will receive the *question(p)* message along with the protocol of figure 4. The agent

identifies the clause which it should use. In this example, the clause playing the ‘provider’ role. It is necessary to use constraints to fully satisfy the second rule. Part of the rule states an agent sending an *assert(p)* depends on its knowledge base and its assertion attitude, otherwise an *assert(null)* is sent. The constraint *verify(p)* is assumed to be satisfiable by the agent. The agent is free to satisfy the constraint how it prefers. This could range from a simple function call to a complex esoteric belief logic with identity evaluation. The protocol only states what conditions must be satisfied, not how. The recursive steps are handled by the roles of *eval* (evaluate) and *def* (defend) which are similarly constrained. Finally, the termination rule for the game is written as the last line in the ‘evaluate’ role. No more messages are sent when the remainder of the set of propositions is empty.

```

agent(infoseek(P, B), A) ::
  agent(provider(P, A), B) or
  agent(seeker(P, B), A).

agent(seeker(P, B), A) ::
  question(P) ⇒ agent(provider(P, A), B) then
  assert(P) ⇐ agent(provider(P, A), B) then
  agent(eval(P, B), A) or
  assert(null) ⇐ agent(provider(P, A), B).

agent(provider(P, A), B) ::
  question(P) ⇐ agent(seeker(P, B), A) then
  (assert(P) ⇒ agent(seeker(P, B), A)
  ⇐ verify(P) then
  agent(def(P, A), B)) or
  assert(null) ⇒ agent(seeker(P, B), A).

agent(eval([P|R], B), A) ::
  accept(P) ⇒ agent(def([P|R], A), B)
  ⇐ accept(P) or
  ( challenge(P) ⇒ agent(def([P|R], A), B) then
    ( assert(S) ⇐ agent(def([P|R], A), B) then
      agent(eval(S, B), A)
    )
  )
  then
  ( null ⇐ R = [] or
    agent(eval(R, B), A) ) .

agent(def([P|R], A), B) ::
  accept(P) ⇐ agent(eval([P|R], B), A) or
  ( challenge(P) ⇐ agent(eval([P|R], B), A) then
    ( assert(S) ⇒ agent(eval([P|R], B), A)
      ⇐ justify(P, S)
    )
  )
  .

```

Figure 4: The information-seeking protocol

Similar protocols can be written to express the other atomic dialogue types. Real world dialogues rarely consist of a single dialogue game type. McBurney and Par-

sons (2002) formally describe several combinations of dialogue types. *Iteration* is the initiation of a dialogue game immediately following the finishing of another dialogue game of the same type. *Sequencing* is the similar to iteration except that the following dialogue game can be of any type. In *Parallelisation* of dialogue games, agents make moves in more than one dialogue game concurrently. *Embedding* of dialogue games occurs when during play of one dialogue game another game is initiated and played to its conclusion before the agents continue playing the first. The example involves two agents; a doctor and a patient. The patient is trying to find out whether the proposition ‘patient is ill’ is true (i.e. looking for a diagnosis). This is the perfect scenario to play an information-seeking dialogue game and to use the dialogue game protocol. Figure 5 and 6 shows the agent clauses as they are rewritten during the course of the dialogue.

$$\begin{aligned} & \text{agent}(\text{infoseek}(\text{"patient is ill"}, \text{doctor}), \text{patient}) :: \\ & \text{agent}(\text{seeker}(\text{"patient is ill"}, \text{doctor}), \text{patient}) \end{aligned} \quad (1)$$

$$\begin{aligned} & \text{agent}(\text{infoseek}(\text{"patient is ill"}, \text{doctor}), \text{patient}) :: \\ & \text{question}(\text{"patient is ill"}) \Rightarrow \\ & \text{agent}(\text{provider}(\text{"patient is ill"}, \text{patient}), \text{doctor}) \end{aligned} \quad (2)$$

$$\begin{aligned} & \text{agent}(\text{infoseek}(\text{"patient is ill"}, \text{doctor}), \text{patient}) :: \\ & \text{question}(\text{"patient is ill"}) \Rightarrow \\ & \text{agent}(\text{provider}(\text{"patient is ill"}, \text{patient}), \\ & \quad \text{doctor}) \text{ then} \\ & \text{assert}(\text{null}) \Leftarrow \\ & \text{agent}(\text{provider}(\text{"patient is ill"}, \text{patient}), \text{doctor}). \end{aligned} \quad (3)$$

Figure 5: The agent clauses for the patient

$$\begin{aligned} & \text{agent}(\text{infoseek}(\text{"patient is ill"}, \text{patient}), \text{doctor}) :: \\ & \text{agent}(\text{provider}(\text{"patient is ill"}, \text{patient}), \text{doctor}) \\ & \quad (4) \\ & \text{agent}(\text{infoseek}(\text{"patient is ill"}, \text{patient}), \text{doctor}) :: \\ & \text{question}(\text{"patient is ill"}) \Leftarrow \\ & \text{agent}(\text{seeker}(\text{"patient is ill"}, \text{doctor}), \\ & \quad \text{patient}) \text{ then} \\ & \text{assert}(\text{null}) \Rightarrow \\ & \text{agent}(\text{seeker}(\text{"patient is ill"}, \text{doctor}), \text{patient}). \end{aligned} \quad (5)$$

Figure 6: The agent clauses for the doctor

The patient begins the dialogue by taking the initial agent clause of *infoseek* which stands for information-seeking. This step is labelled 1. The agent applies the rewrite rules to expand the seeker role and sends the *question* to the doctor agent, step 2. The doctor receives the message and the protocol. The applies the rewrite rules and finds the only instantiation that is possible is the unfolding of the provider role. It applies the rewrite rules and comes to the *verify* constraint which it is unable to satisfy. It cannot determine the truth value of the proposition and is unwilling to defend the proposition. It takes the other half of the *or* operator and sends the *assert(null)*. Let us assume the doctor agent is a bit more clever. It cannot currently assert that the patient is ill. It has a knowledge-base and an inference engine that allows it to figure whether the proposition is true or not, and it needs some more information from the patient. The particular kind of information would depend on each patient consultation. If this diagnosis scenario was part of an electronic institution, the institution would have to represent in a state diagram every possible permutation of a diagnosis scenario. This is not practical, if not impossible.

Instead, the doctor agent can use the patterns of dialogue games to structure the interaction but allow adaptations to handle any run-time dialogical needs that may arise. In the example, the doctor agent needs to ask about a different proposition before it can answer the patient’s original query. This is achieved by an additional rewrite rule shown in figure 7.

$$\begin{aligned} & A \xrightarrow{M_i, M_o, P, O} A \text{ then } B \\ & \text{if } \text{clause}(P, B) \wedge \text{isa}(B, \text{dialogue} - \text{type}) \\ & \text{isa}(\text{infoseek}, \text{dialogue} - \text{type}). \end{aligned}$$

Figure 7: Additional rewrite rule

This allows the agent to graft the *infoseek* agent clause between any term in the protocol. These rewrites can be expanded further to represent other dialogue combinations as well as domain specific rewrite rules. The doctor’s dialogue clause with the use of the embedding is shown in figure 8. The expansions and dialogue begin the same, but rather than just sending the *assert(null)*. The agent inserts the agent definition *agent(infoseek("patient has a fever"), patient), doctor)*. The next instance of an information-seeking dialogue is begun. The moves of the embedded dialogue game are in bold text. In this instance the patient plays the provider role and the doctor plays the seeker. The game is finished by the patient asserting "patient has a fever". The doctor, now knowing this proposition to be true, has enough knowledge to assert the original proposition posed by the patient’s first question. The first information-seeking game also concludes successfully by the doctor making the diagnosis and asserting the proposition "patient is ill" is true.

```

agent(infoseek("patient is ill", patient), doctor) ::
question("patient is ill") ⇐
agent(seeker("patient is ill", doctor), patient) then
agent(infoseek("patient has a fever", patient)
      , doctor).

```

(6)

```

agent(infoseek("patient is ill", patient), doctor) ::
question("patient is ill") ⇐
agent(seeker("patient is ill", doctor), patient) then
question("patient has a fever") ⇒
agent(provider("patient has a fever", doctor)
      , patient)

```

(7)

...

```

agent(infoseek("patient is ill", patient), doctor) ::
question("patient is ill") ⇐
agent(seeker("patient is ill", doctor), patient) then
question("patient has a fever") ⇒
agent(provider("patient has a fever", doctor)
      , patient) then
assert("patient has a fever") ⇐
agent(provider("patient has a fever", doctor)
      , patient) then
assert("patient is ill") ⇒
agent(seeker("patient is ill", doctor), patient).

```

Figure 8: Embedded information-seeking agent clause for doctor

6 Conclusions

The protocol language described in the paper is expressive enough to represent the most popular approaches to the agent communication. It is able to capture the various aspects of Electronic Institutions such as the scenes, performative structure, and normative rules. This enables agents to have structured and meaningful dialogues without relying on centralised control of the conversation. The language is also capable of facilitating agent-centric approaches to agent communication. Agents pass the protocol to their dialogical partners to communicate the social conventions for the interaction. Agents can adapt the received protocols to explore dynamic conversation spaces. The protocol language in this paper is not seen as a replacement for either model of agent communication. Instead, it synthesises the two approaches to gain the advantages of both. Protocols are used to coordinate and guide the agent's dialogue, but agents are able to adapt the protocol by using an agent-centric model for communication. The use of this communicative model constrains transformation to the agent clauses in meaningful ways.

The run-time delivery provides the mechanism for communicating the protocol as well as any adaptations that are made. We have begun developing a FIPA compliant agent which uses the ACL library and the protocol language. It is hoped that the verifiability and semantic problems associated with FIPA's ACL can be mitigated by the use of the protocol language to communicate the performative's semantics during their use.

This approach does raise new issues which have not been addressed in this paper. One issue concerns restricting changes to the protocols. There are certainly dialogues where certain agents will be restricted from modifying the protocols or dialogue which require portions of the protocol to remain unchanged. This remains for future work along with development of a vocabulary of generic transformations which can be proven *a priori* or verified to retain semantic and syntactical continuity of the protocols.

The protocol language has already been shown to be useful for a number of agent purposes. A scheduling program has been developed using the protocol written in Prolog and using LINDA. A Java-based agent framework also exists which uses an XML representation of the protocols. Separating the protocol from the deliberative and communicative models of agency makes definition and verification simpler tasks. Tools have already been developed which use model-checking for automatic verification Walton (2004a). The protocol language has been used to implement the generic dialogue framework of McBurney and Parsons (2002) and the negotiation game described in McBurney et al. (2002).

References

- Marc Esteva, Juan A. Rodriguez, Carles Sierra, Pere Garcia, and Josep L. Arcos. On the formal specifications of electronic institutions. *LNAI*, pages 126–147, 2001.
- Marc Esteva, David de la Cruz, and Carles Sierra. Islander: an electronic institutions editor. In *Proceeding of the first International joint conference on Autonomous agents and multiagent systems*, pages 1045–1052, Bologna, Italy, 2002. ACM press.
- Marc Esteva, Juan A. Rodriguez-Aguilar, Josep Ll. Arcos, Carles Sierra, and Pere Garcia. Institutionalising open multi-agent systems. In *proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS'2000)*, pages 381–83, Boston, 2000. ICMA.
- FIPA. FIPA communicative act library specification, <http://www.fipa.org/specs/fipa00037/XC00037H.html>, 2001.
- Robert A. Flores and R.C. Kremer. To commit or not to commit: Modelling agent conversations for action. *Computational Intelligence*, 18(2):120–173, May 2002.

- Foundation for Intelligent Physical Agents. Fipa sl content language specification, 2000.
- Mark Greaves, Heather Holmback, and Jeffrey Bradshaw. What is a conversation policy? In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pages 118–131. Springer-Verlag: Heidelberg, Germany, 2000.
- Nicolas Maudet and Brahim Chaib-draa. Commitment-based and dialogue-game based protocols: new trends in agent communication languages. *The Knowledge Engineering Review*, 17(2), 2002.
- Nicolas Maudet and Fabrice Evrard. A generic framework for dialogue game implementation, 1998.
- Peter McBurney and Simon Parsons. Games that agents play: A formal framework for dialogues between autonomous agents. *Journal of Logic, Language and Information*, 11(3):315–334, 2002.
- Peter McBurney, Rogier van Eijk, Simon Parsons, and Leila Amgoud. A dialogue-game protocol for agent purchase negotiations. *Journal of Autonomous Agents and Multi-Agent Systems*. (In press), 2002.
- Jarred McGinnis, David Robertson, and Chris Walton. Using distributed protocols as an implementation of dialogue games. Presented EUMAS 2003, December 2003.
- Simon Parsons, Peter McBurney, and Michael Wooldridge. The mechanics of some formal inter-agent dialogues. In *Workshop on Agent Communication Languages*, pages 329–348, 2003.
- Philippe Pasquier, Nicolas Andriillon, and Brahim Chaib-draa. An exploration in using the cognitive coherence theory to automate agents’s communicational behavior. In *Agent Communication Language and Dialogue workshop*, Melbourne, Australia, 2003. AAMAS’03.
- Chris Reed. Dialogue frames in agent communication. In Y. Demazeau, editor, *Proceedings of the Third International Conference on Multi-Agent Systems(ICMAS-98)*, pages 246–253. IEEE Press, 1998.
- David Robertson. A lightweight coordination calculus for agent social norms. In *3rd International Conference on Autonomous Agents and Multi Agent Systems*, New York, USA, 2004a. paper in submission, available from author.
- David Robertson. A lightweight method for coordination of agent oriented web services. In *Proceedings of AAAI Spring Symposium on Semantic Web Services*, California, USA, 2004b.
- John Searle. *Speech Acts*. Cambridge University Press, 1969.
- Munindar P. Singh. A social semantics for agent communication languages. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pages 31–45. Springer-Verlag: Heidelberg, Germany, 2000.
- Wamberto Vasconcelos. Skeleton-based agent development for electronic institutions. In *First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Bologna, Italy, July 2002. AAMAS, ACM Press.
- Chris Walton and Dave Robertson. Flexible multi-agent protocols. Technical Report EDI-INF-RR-0164, University of Edinburgh, 2002.
- Chris D. Walton. Model Checking Multi-Agent Web Services. In *Proceedings of the 2004 AAAI Spring Symposium on Semantic Web Services (To Appear)*, Stanford, California, March 2004a.
- Chris D. Walton. Multi-Agent Dialogue Protocols. In *Proceedings of the Eighth International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida, January 2004b.
- Doug Walton and Eric C. W. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. SUNY press, Albany, NY, USA, 1995.
- Michael Wooldridge. Semantic issues in the verification of agent communication languages. *Autonomous Agents and Multi-Agent Systems*, 3(1):9–31, 2000.

Robust Online Reputation Mechanism by Stochastic Approximation

Takamichi Sakai*

*NTT Communication Science Laboratories,
NTT Corporation
2-4 Hikaridai, Seika-cho, Soraku-gun, Kyoto,
619-0237 Japan
sakai@cslab.kecl.ntt.co.jp

Kenji Terada†

†NTT East Research and Development Center,
NTT East Corporation
3-19-2 Nishi-shinjuku, Shinjuku-ku, Tokyo,
163-8019 Japan
kenji.terada@rdc.east.ntt.co.jp

Tadashi Araragi‡

‡NTT Communication Science Laboratories,
NTT Corporation
2-4 Hikaridai, Seika-cho, Soraku-gun, Kyoto,
619-0237 Japan
araragi@cslab.kecl.ntt.co.jp

Abstract

Recently, online reputation mechanisms have attracted much attention in many areas. They have been widely adopted and worked well, although their reliability is still a major concern. Because of online properties such as openness and anonymity, it is necessary to consider rating errors, noise and unfair lies. Furthermore, these disturbances (attacks) have a significant effect on multi-agent systems containing malicious agents who tell lies or engage in strategic manipulations. Current online reputation mechanisms are not sufficiently robust against such disturbances. In an attempt to solve this problem, in this paper we propose a stochastic approximation-based online reputation mechanism. Our mechanism assigns one global trustworthiness value to each agent and updates estimates of these values dynamically from mutual ratings of agents. Experimental results show that our mechanism is able to identify good and bad agents effectively under condition of the above disturbances and also trace the changes in agents' true trustworthiness values adaptively.

1 Introduction

Recently online reputation mechanisms have attracted much attention in many areas, such as multi-agent systems, peer-to-peer systems, and electronic commerce. In such areas, there is an enormous number of unfamiliar potential trading partners, so *reputation information* of partners can take a central role in selecting partners to contact. Here, reputation information is a collection of many users' ratings about trustworthiness of the specified partner. The role of reputation information is larger for an agent than for human, because an agent has difficulty in feeling the partner's trustworthiness from the partner's advertising statements, responses and so on. Furthermore, an online reputation mechanism is more cost-effective than establishing an authentic oversight mechanism like a law-enforcement agency (Kollock, 1999).

The basic mechanism of an online reputation mechanism is that users rate each other, and their ratings (i.e. reputation information) become publicly available (Delarocas, 2003). For example, the mechanism of the major Internet auction site eBay (www.ebay.com) is as follows. When a transaction between a seller and a buyer finishes, the seller and the buyer rate each other: they choose one level from positive, neutral or negative, and they can add a comment, too. These ratings are subsequently collected

and made publicly available. Each user's reputation information consists of a table and a sheet. The table shows the total counts of each rating level that the user has acquired during past 1, 6 and 12 months, and the sheet lists ratings the user acquired in the newest-first order, where each rating consists of a rating level, a comment and a rater. In this way each user can estimate the trustworthiness of an unfamiliar partner by referring to the partner's reputation information. Furthermore, the effect of suppressing a user's bad behaviors can be expected because they receive bad ratings in feedback when they behave badly.

Online reputation mechanisms have been widely adopted and worked well, although their reliability is still a major concern. Because of online properties such as openness and anonymity, the online reputation mechanism has to be robust against rating errors, noise and unfair lies. Furthermore, these disturbances (attacks) by malicious agents or groups of them, who tell lies or engage in strategic manipulations, have a significant effect on multi-agent systems where human do not intervene. Current online reputation mechanisms are not sufficiently robust against such disturbances, so developing a secure online reputation mechanism remains a significant challenge in multi-agent systems.

In this paper, we propose a stochastic approximation-

based online reputation mechanism in multi-agent systems. Our mechanism assigns one global trustworthiness value to each agent as its reputation information and updates estimates of this value dynamically from mutual ratings of agents. Experimental results show that our mechanism is able to identify good and bad agents effectively under conditions of the above disturbances. Moreover, our mechanism is able to trace the changes in agents' true trustworthiness values adaptively, which is the situation where agents with good behaviors suddenly start performing manipulations.

This paper's structure is as follows. First, we discuss works related to online reputation mechanisms and present the position of this paper in 2. Then, we show our framework of the online reputation mechanism and define the problem in 3. We then propose our solution to the problem, stochastic approximation-based algorithms, in 4, and provide experimental results of these algorithms in 5. Finally, we conclude in 6.

2 Related Work

There are many related works about online reputation mechanisms. In this chapter we focus on the related works concerning mechanism design that propose novel mechanisms or algorithms in terms of robustness or other aspects. Furthermore, we clarify the position of this paper by classifying related works from the perspective of the representation of trust (2.1) and the subjectivity of trust (2.2). By the way, for clarity, let Agent A 's trust in Agent B (B 's trustworthiness for A) be $T(A, B)$, from now on.

2.1 Representation of Trust

When designing online reputation mechanisms, we have to decide the representation of trust $T(A, B)$ from the beginning. If its user is human, the representation of trust $T(A, B)$ is possible by using natural language. However, if its user is an agent, a computer-friendly representation format of trust is suitable. In this paper and Ishida (1996); Zacharia et al. (1999); Dellarocas (2000), trust $T(A, B)$ is represented by one (real or natural) number. The bigger this value is, the more A trusts B . Basically this representation can include eBay's ratings of three levels, except for comments. This representation is easy to handle, although it does feature a limitation. For example, this representation cannot handle trust in the manner that "I can trust her about computers, but I cannot trust her about cooking," or that "the seller provides high quality goods but the seller's delivery is late." In general, trust should be rated by many viewpoints, and in order to represent such detailed trust, more complex representation formats like vectors are suitable. Indeed, as a representative format of trust, Yu and Singh (2003) uses the Dempster-Shafer theory-based basic probability assignment, Wang and Vassileva (2003) uses Bayesian network.

2.2 Subjectivity of Trust

One major role of the online reputation mechanism is estimating the trustworthiness of an unfamiliar agent. There are main two approaches to this estimation. We explain the two approaches using Figure 1. Figure 1 shows the

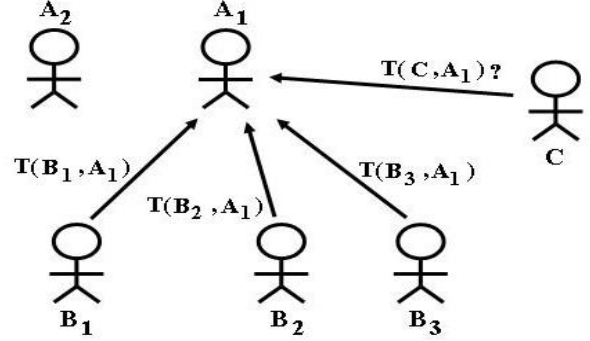


Figure 1: Trustworthiness estimation of an unfamiliar agent.

situation where Agent C wants to know the Agent A_1 's trustworthiness $T(C, A_1)$ in the existence of Agents A_1 , A_2 , B_1 , B_2 , B_3 and C . For example, this is a situation where A_1 and A_2 are sellers providing a similar service, and buyer C wants to know the trustworthiness of A_1 and A_2 to decide which seller C receives the service from. If C traded with A_1 directly in the past, C can estimate $T(C, A_1)$ from C 's experiences by itself. Here, however, it is assumed that A_1 is unfamiliar with C . In this case, C receives trustworthiness of A_1 from other agents. Here, we assume B_1 , B_2 , B_3 traded with A_1 directly in the past and has the rating $T_1 = T(B_1, A_1)$, $T_2 = T(B_2, A_1)$, $T_3 = T(B_3, A_1)$, respectively. Then C can estimate trustworthiness $T(C, A_1)$ by integrating T_1 , T_2 and T_3 . However, the method of integration differs by two approaches.

The first approach (this paper and Ishida (1996); Zacharia et al. (1999)) assumes the existence of a global trustworthiness $T_g(A_1) = T(\text{agency}, A_1)$ for each agent. Here, $T_g(A_1)$ represents the trustworthiness of A_1 for agency (all agents). $T_g(A_1)$ is an objective and absolute rating; that is, the trustworthiness of A_1 for B_1 is the same as for B_2 , and for every different agent. $T_g(A_1)$ is computed by integrating (e.g. averaging) all agents' trust in A_1 , and C uses this $T_g(A_1)$ as the estimation of $T(C, A_1)$. Although the assumption of this value is arguable, this approach has a merit in that a newly arriving agent can use this value $T_g(A_1)$ immediately to estimate the trustworthiness of an unfamiliar agent (here, A_1).

The second approach (Dellarocas, 2000; Yu and Singh, 2003; Wang and Vassileva, 2003) does not assume the existence of a global trustworthiness $T_g(A_1)$. Instead, this approach considers that trust is personal, subjective, and relative, not global. That is, the trustworthiness of A_1 for B_1 is different from that for B_2 , and for every different agent. Therefore, in order to estimate $T(C, A_1)$, we

should search agents whose ratings or preference is similar to C . Here, we assume B_2 and B_3 are searched as the similar agents to C . Then the estimation of $T(C, A_1)$ is computed by integrating only these trusts T_2 and T_3 . Although this approach requires a reasonable measure of similarity between agents, the trust value can be relatively meaningful.

2.3 Position of This Paper

This paper use one real number as a trust representation and assumes the existence of a global trustworthiness.

Some related (Ishida, 1996; Zacharia et al., 1999) works have already adopted the same framework as this paper. However, they (Ishida, 1996; Zacharia et al., 1999) derive estimation algorithms of trustworthiness values in an ad-hoc style, and their theoretical support is not strong enough (see 4.1.3). This paper presents stochastic approximation-based algorithms and in particular, analytical and experimental results about robustness against disturbances.

3 Online Reputation Mechanism

In this chapter, we show the online reputation mechanism that we assume, and define the problem.

We assume an Internet-based multi-agent system where there is an enormous number of autonomous agents interacting with each other, and some agents may enter or leave in real time. When Agent j obtains a service from Agent i , Agent j rates Agent i , for instance, on the basis of the quality of Agent i 's service, and reports the rating value $R_{ji} \in [0, 1]$ to the online reputation mechanism (see Figure 2), where $R_{ji} = 1(0)$ means that Agent i is quite good (bad). R_{ji} should be treated as a stochastic variable, since there exist rating errors and noise. In addition, the value of R_{ji} is not always true, since malicious agents may tell lies as a tactic of manipulation. Furthermore, the online reputation mechanism assigns a global trustworthiness value of each Agent k ($k = 1, \dots, r$)¹, and has its estimates $\theta_k \in [0, 1]$. Here, $\theta_k = 1(0)$ means that Agent k is quite good, honest (bad, malicious). Then, the online reputation mechanism provides this estimate θ_k when it receives inquiry of Agent k 's trustworthiness.

The problem of the online reputation mechanism, therefore, is how to estimate the trustworthiness values $\theta = (\theta_1, \dots, \theta_r)$, which converge to the true trustworthiness values $\bar{\theta} = (\bar{\theta}_1, \dots, \bar{\theta}_r)$, dynamically from the sequence of the reports $\{R_{ji}\}$, especially where $\{R_{ji}\}$ contains rating errors, noise and unfair lies by malicious agents.

Note that in our framework an agent's trustworthiness is represented by only one real number, even though it has two meanings. That is, $\bar{\theta}_k$ represents both the trustworthiness of Agent k 's "service" and the trustworthiness

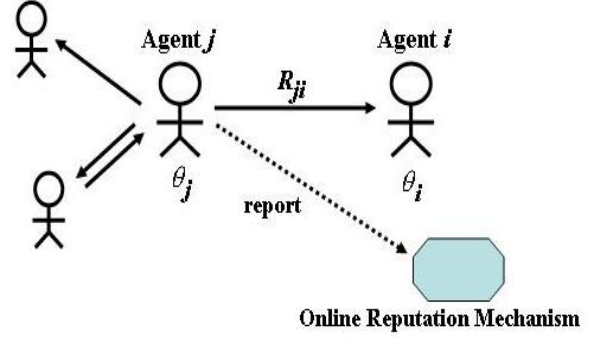


Figure 2: Framework of the online reputation mechanism.

of Agent k 's "rating" for other agents (i.e. whether Agent k is a liar or not). Strictly speaking, these two trustworthiness factors are independent. For example, there can exist agents whose "service" is good, but whose "rating" is bad (that is, they are liars). However, these two trustworthiness factors are identical in many cases, and in addition, the needs of users are knowing agents that offer both a good "service" and "rating." Hence we define the global trustworthiness value $\bar{\theta}_k$ as a conjunction (AND, \wedge) of these two trustworthiness factors. That is to say, in our framework, a good agent ($\bar{\theta}_k = 1$) means that both Agent k 's "service" and "rating" are good, and a bad agent ($\bar{\theta}_k = 0$) means either Agent k 's "service" or "rating" is bad, or both are bad. Conversely, we are unable to distinguish, for instance, whether $\bar{\theta}_k = 0$ indicates that Agent k 's "service" is bad, Agent k 's "rating" is bad, or both. To handle this type of trustworthiness precisely, we have to employ an even more complex representation of trust, such as vectors (see 2.1). However, we consider our simple representation by one real number to usually be sufficient for many applications.

4 Stochastic Approximation-Based Algorithm

In this chapter, we provide three algorithms that update the trustworthiness estimates θ recursively from the mutual ratings $\{R_{ji}\}$. Each algorithm is obtained in a similar fashion from the corresponding rating model. We explain about asynchronous algorithms suitable for practical use, and introduce the adaptive algorithm to adjust step-size parameters automatically.

4.1 Stochastic Gradient Descent Algorithm

4.1.1 Rating Model

To obtain the algorithms, we set a *rating model* first. The rating model is a function $u(\bar{\theta})$ describing our expectations of how R_{ji} is generated, i.e. we define the expectation $E[R_{ji}] = u_{ji}(\bar{\theta})$. In this paper, we use the following

¹ r is the number of agents and also dimensions of θ .

three typical rating models. However, these three models are just examples, and the following developments hold true even when using other rating models.

$$\text{LLS} \quad u_{ji}^{\text{LLS}}(\bar{\theta}) = \bar{\theta}_i, \quad (1)$$

$$\text{Joint} \quad u_{ji}^{\text{Joint}}(\bar{\theta}) = \bar{\theta}_j \bar{\theta}_i, \quad (2)$$

$$\text{Lie} \quad u_{ji}^{\text{Lie}}(\bar{\theta}) = 2\bar{\theta}_j \bar{\theta}_i - \bar{\theta}_j - \bar{\theta}_i + 1^2. \quad (3)$$

The *Linear Least-Squares (LLS)* model is the simplest model, which predicts that R_{ji} reflects the true trustworthiness value $\bar{\theta}_i$ on average and will lead to a linear least-squares estimate of $\bar{\theta}_i$. The *Joint* model assumes that the transaction between Agent j and Agent i will succeed ($R_{ji} = 1$) if both Agents j and i are good ($\bar{\theta}_j = 1 \wedge \bar{\theta}_i = 1$). The *Lie* model deals with lying agents that always express the complementary (NOT, \neg) rating. Indeed, when rater Agent j 's trustworthiness is $\bar{\theta}_j = 0$, this model expects the rating R_{ji} to be $1 - \bar{\theta}_i$, which is complementary of Agent i 's true trustworthiness value $\bar{\theta}_i$.

4.1.2 Derivation of Algorithms

Next, to estimate θ , we select the mean square error (MSE) criterion as a cost function to minimize. Here we define

$$\text{MSE} = E \left[\sum_{i,j(i \neq j)} (R_{ji} - u_{ji}(\theta))^2 \right]. \quad (4)$$

That is, we treat R_{ji} as desired outputs and adjust θ to output these desired outputs. In addition, here we assume many R_{ji} are obtained synchronously, and we imply that the summation $\sum_{i,j(i \neq j)}$ is done only by the set of R_{ji} obtained in each synchronization round (we mention an asynchronous case at 4.2). We also define the *sample* MSE

$$e(\theta, R) = \frac{1}{2} \sum_{i,j(i \neq j)} (R_{ji} - u_{ji}(\theta))^2 \quad (5)$$

and the *sample* error

$$e_{ji} = R_{ji} - u_{ji}(\theta). \quad (6)$$

The gradient values of the MSE are not known, although these “noise-corrupted” observations can be taken. Then we can use the stochastic approximation (Robbins and Monro) method (Kushner and Yin, 2003) to obtain this *stochastic gradient descent algorithm*:

$$\begin{aligned} \theta'_k &= \theta_k - \epsilon_k \frac{\partial e(\theta, R)}{\partial \theta_k} \\ &= \theta_k + \epsilon_k \sum_{i,j(i \neq j)} e_{ji} \frac{\partial u_{ji}(\theta)}{\partial \theta_k}, \quad k=1, \dots, r, \end{aligned} \quad (7)$$

²A more general form is $(1 - 2\bar{\theta}_i)(1 - \bar{\theta}_j)^a + \bar{\theta}_i$, which leads to (3) when $a = 1$.

where θ'_k denotes the next time step's value of θ_k , and ϵ_k is the k th component of the *step-size parameter (learning rate)*. If ϵ_k 's sequence $\{\epsilon_{k,n} (n = 1, \dots)\}$ satisfies

$$\epsilon_{k,n} > 0, \quad \epsilon_{k,n} \rightarrow 0, \quad \sum_n \epsilon_{k,n} = \infty, \quad (8)$$

θ is converged to the minimum point with the probability of one (w.p.1).

Substituting $u_{ji}(\theta)$ in (7) by the models $u_{ji}^{\text{LLS}}(\theta)$, $u_{ji}^{\text{Joint}}(\theta)$ and $u_{ji}^{\text{Lie}}(\theta)$, we obtain concrete algorithms for each respective model:

$$\text{LLS} \quad \theta'_k = \theta_k + \epsilon_k \sum_{j(j \neq k)} (R_{jk} - \theta_k), \quad (9)$$

$$\text{Joint} \quad \theta'_k = \theta_k + \epsilon_k \sum_{j(j \neq k)} \theta_j ((R_{jk} - \theta_j \theta_k) + (R_{kj} - \theta_k \theta_j)), \quad (10)$$

$$\text{Lie} \quad \theta'_k = \theta_k + \epsilon_k \sum_{j(j \neq k)} (2\theta_j - 1)((R_{jk} - 2\theta_j \theta_k + \theta_j + \theta_k - 1) + (R_{kj} - 2\theta_k \theta_j + \theta_k + \theta_j - 1)). \quad (11)$$

Here we supplement one item. The domain of θ_k is constrained by $[0, 1]$; therefore, equation (7) is used by the following *projected* form in practice,

$$\theta'_k = \prod_{[0,1]} \left[\theta_k + \epsilon_k \sum_{i,j(i \neq j)} e_{ji} \frac{\partial u_{ji}(\theta)}{\partial \theta_k} \right], \quad (12)$$

where $\prod_H(\theta)$ denotes the closest point in H to θ . If θ is one-dimensional and H is $[a, b]$, $\prod_H(\theta)$ is the same as

$$\prod_{[a,b]}(\theta_k) = \min(\max(\theta_k, a), b). \quad (13)$$

We do not indicate this clearly just for simplicity: we always assume that the right side of θ_k 's updating equations including (9), (10), and (11) has the projected form by $\prod_{[0,1]}$ implicitly.

4.1.3 Discussion of Other Algorithms

(9) shows that the finite difference in the estimate θ_k can be interpreted as the sum of the product of the “estimation error” $R_{jk} - \theta_k$ and the coefficient of “reliability” ϵ_k (Kushner and Yin, 2003). Therefore, we can derive another algorithm by incorporating the rater j 's trustworthiness value θ_j into the coefficient of “reliability”:

$$\text{Weighted} \quad \theta'_k = \theta_k + \epsilon_k \sum_{j(j \neq k)} \theta_j (R_{jk} - \theta_k). \quad (14)$$

This *Weighted* algorithm forms the basis of the algorithms used in the related works (Ishida, 1996; Zacharia et al., 1999). However, the cost function of the *Weighted* algorithm is not clear, and we can derive more effective algorithms as described below.

First, we consider a more general MSE than (4):

$$\text{MSE}^\dagger = E \left[\sum_{i,j(i \neq j)} w_{ji} (R_{ji} - u_{ji}(\theta))^2 \right], \quad (15)$$

where w_{ji} denotes the coefficient of each square error's weight. We treat all square errors' weights as an equal ($w_{ji} = 1$) in (4). Now, however, we place a special weight (influence) on square errors' weights associated with raters of high trustworthiness, i.e. we set $w_{ji} = \theta_j$ here³. We also define the sample MSE^\dagger

$$e^\dagger(\theta, R) = \frac{1}{2} \sum_{i,j(i \neq j)} w_{ji} (R_{ji} - u_{ji}(\theta))^2, \quad (16)$$

and employ the stochastic approximation method to obtain this stochastic gradient descent algorithm:

$$\begin{aligned} \theta'_k &= \theta_k - \epsilon_k \frac{\partial e^\dagger(\theta, R)}{\partial \theta_k} \\ &= \theta_k + \epsilon_k \sum_{i,j(i \neq j)} \left(w_{ji} e_{ji} \frac{\partial u_{ji}(\theta)}{\partial \theta_k} - \frac{1}{2} e_{ji}^2 \frac{\partial w_{ji}}{\partial \theta_k} \right) \\ &= \theta_k + \epsilon_k \sum_{i,j(i \neq j)} \left(\theta_j e_{ji} \frac{\partial u_{ji}(\theta)}{\partial \theta_k} - \frac{1}{2} e_{ji}^2 \delta_{j,k} \right), \end{aligned} \quad (17)$$

$k = 1, \dots, r$. Here, $\delta_{j,k}$ is the Kronecker delta, which becomes 1 when $j = k$ and 0 when $j \neq k$. In addition, we used $w_{ji} = \theta_j$ to change the equation of the second line to that of the third line in (17).

Substituting $u_{ji}(\theta)$ in (17) by the model $u_{ji}^{\text{LLS}}(\theta)$, we obtain the concrete algorithm⁴:

$$\begin{aligned} \text{LLS}^w \quad \theta'_k &= \theta_k + \epsilon_k \sum_{j(j \neq k)} (\theta_j (R_{jk} - \theta_k) - \\ &\quad \frac{1}{2} (R_{kj} - \theta_j)^2). \end{aligned} \quad (18)$$

Comparing it with the Weighted algorithm (14), we see that the only difference between (14) and this LLS^w algorithm (18) is the second term in \sum . This term indicates the error between the Agent k 's rating of another Agent j , and Agent j 's trustworthiness value θ_j is reflected to Agent k 's own trustworthiness value θ_k . For this reason, we can interpret this term as the reflection effect as Ishida (1996) mentioned. This term enables the LLS^w algorithm to perform better than the Weighted algorithm (see 5.2).

4.2 Asynchronous Algorithm

In the preceding section 4.1, we dealt with the synchronous case in which many R_{ji} are synchronously obtained. There exist many applications working synchronously, like a sensor network (Ishida, 1996). However, the online reputation mechanism framework of this

paper assumes that each R_{ji} is obtained asynchronously. In this section, we derive asynchronous algorithms and discuss problems like the order and frequency of updating, which arise in the asynchronous case.

4.2.1 Derivation of Asynchronous Algorithms

We estimate θ by minimizing this MSE^* in the asynchronous case.

$$\text{MSE}^* = E \left[w_{ji} (R_{ji} - u_{ji}(\theta))^2 \right]. \quad (19)$$

We also define the sample MSE^*

$$e^*(\theta, R_{ji}) = \frac{1}{2} w_{ji} (R_{ji} - u_{ji}(\theta))^2, \quad (20)$$

and use the stochastic approximation method to obtain this stochastic gradient descent algorithm:

$$\begin{aligned} \theta'_k &= \theta_k - \epsilon_k \frac{\partial e^*(\theta, R_{ji})}{\partial \theta_k} \\ &= \theta_k + \epsilon_k \left(w_{ji} e_{ji} \frac{\partial u_{ji}(\theta)}{\partial \theta_k} - \frac{1}{2} e_{ji}^2 \frac{\partial w_{ji}}{\partial \theta_k} \right), \end{aligned} \quad (21)$$

$k = 1, \dots, r$. However, both $u_{ji}(\theta)$ and w_{ji} are usually the function of θ_i and θ_j only. In this case,

$$\frac{\partial u_{ji}(\theta)}{\partial \theta_k} = \frac{\partial w_{ji}}{\partial \theta_k} = 0 \quad (k \neq i \wedge k \neq j),$$

and the equation (21) is expanded as follows:

$$\theta'_i = \theta_i + \epsilon_i \left(w_{ji} e_{ji} \frac{\partial u_{ji}(\theta)}{\partial \theta_i} - \frac{1}{2} e_{ji}^2 \frac{\partial w_{ji}}{\partial \theta_i} \right), \quad (22)$$

$$\theta'_j = \theta_j + \epsilon_j \left(w_{ji} e_{ji} \frac{\partial u_{ji}(\theta)}{\partial \theta_j} - \frac{1}{2} e_{ji}^2 \frac{\partial w_{ji}}{\partial \theta_j} \right), \quad (23)$$

$$\theta'_k = \theta_k \quad (k \neq i \wedge k \neq j). \quad (24)$$

That is, we have to update at most two components i and j at each R_{ji} , and need not update (sweep) all components of θ .

Substituting $u_{ji}(\theta)$ in (22)–(24) by the models $u_{ji}^{\text{LLS}}(\theta)$, $u_{ji}^{\text{Joint}}(\theta)$ and $u_{ji}^{\text{Lie}}(\theta)$, we obtain concrete algorithms for each respective model (in the case of $w_{ji} = 1$):

$$\text{LLS} \quad \theta'_i = \theta_i + \epsilon_i (R_{ji} - \theta_i), \quad (25)$$

$$\text{Joint} \quad \theta'_i = \theta_i + \epsilon_i \theta_j (R_{ji} - \theta_j \theta_i), \quad (26)$$

$$\theta'_j = \theta_j + \epsilon_j \theta_i (R_{ji} - \theta_j \theta_i), \quad (27)$$

$$\begin{aligned} \text{Lie} \quad \theta'_i &= \theta_i + \epsilon_i (2\theta_j - 1) (R_{ji} - 2\theta_j \theta_i + \\ &\quad \theta_j + \theta_i - 1), \end{aligned} \quad (28)$$

$$\begin{aligned} \theta'_j &= \theta_j + \epsilon_j (2\theta_i - 1) (R_{ji} - 2\theta_j \theta_i + \\ &\quad \theta_j + \theta_i - 1). \end{aligned} \quad (29)$$

Furthermore, in the case of $w_{ji} = \theta_j$:

$$\text{LLS}^w \quad \theta'_i = \theta_i + \epsilon_i \theta_j (R_{ji} - \theta_i), \quad (30)$$

$$\theta'_j = \theta_j - \frac{1}{2} \epsilon_j (R_{ji} - \theta_i)^2, \quad (31)$$

³We can use a step function or a logistic function also.

⁴We omit concrete algorithms of Joint^w and Lie^w

$$\text{Joint}^w \quad \theta'_i = \theta_i + \epsilon_i \theta_j^2 (R_{ji} - \theta_j \theta_i), \quad (32)$$

$$\theta'_j = \theta_j + \epsilon_j (\theta_j \theta_i (R_{ji} - \theta_j \theta_i) - \frac{1}{2} (R_{ji} - \theta_j \theta_i)^2), \quad (33)$$

$$\text{Lie}^w \quad \theta'_i = \theta_i + \epsilon_i \theta_j (2\theta_j - 1) (R_{ji} - 2\theta_j \theta_i + \theta_j + \theta_i - 1), \quad (34)$$

$$\theta'_j = \theta_j + \epsilon_j (\theta_j (2\theta_i - 1) (R_{ji} - 2\theta_j \theta_i + \theta_j + \theta_i - 1) - \frac{1}{2} (R_{ji} - 2\theta_j \theta_i + \theta_j + \theta_i - 1)^2). \quad (35)$$

4.2.2 Discussion about Asynchronous Algorithms

In the asynchronous case, the strategy of updating is important for the effective performance of the algorithms, just as asynchronous Dynamic Programming (DP) or Q-learning (Sutton and Barto, 1998) is. If exactly the same set of ratings R_{ji} are given, the difference between synchronous algorithms and asynchronous algorithms is that synchronous algorithms update θ from this set of ratings R_{ji} all at once, whereas asynchronous algorithms update θ many times as each rating R_{ji} is given, where updated θ are used at each update. Therefore, we can ignore this difference if the step-size parameters are adequately small. However, there is the possibility that the ratio of unfair ratings by malicious agents increases in the asynchronous case because malicious agents may wage an intensive attack in a short period of time. Thus, we have to constrain the frequency of each agent's rating report per specified time interval.

4.3 Adaptive Algorithm

When we use the stochastic approximation-based algorithms, the choice of the sequence $\{\epsilon_{k,n}\}$ is an important issue (Kushner and Yin, 2003). For the convergence of θ , (8) is required; however, to track the time-varying parameter $\bar{\theta}$, we usually use the fixed $\epsilon_{k,n} = \epsilon$. In general, if the $\bar{\theta}$ changes faster, ϵ should be larger, and if the observation noise is greater, ϵ should be smaller, though the optimal value of ϵ is unknown in many cases. Here we adopt a useful approach suggested by Benveniste et al. (1990). The idea is to use the stochastic approximation method again to estimate the correct step-size parameter ϵ . By differentiating $e(\theta, R)$ ⁵ with respect to ϵ_k , we obtain the stochastic gradient descent algorithm of ϵ_k :

$$\begin{aligned} \epsilon'_k &= \prod_{[\epsilon_-, \epsilon_+]} \left[\epsilon_k - \mu \frac{\partial e(\theta, R)}{\partial \epsilon_k} \right] \\ &= \prod_{[\epsilon_-, \epsilon_+]} \left[\epsilon_k + \mu \sum_{i,j(i \neq j)} e_{ji} \frac{\partial u_{ji}(\theta)}{\partial \theta_k} V_k \right], \quad (36) \end{aligned}$$

⁵Although we use $e(\theta, R)$ here, we can derive adaptive algorithms of $e^\dagger(\theta, R)$ and $e^*(\theta, R_{ji})$ in a similar fashion.

$k = 1, \dots, r$, where ϵ'_k denotes the next time step's value of ϵ_k , ϵ_- and ϵ_+ are coefficients to constrain the domain of ϵ_k , and μ is the step-size parameter. Additionally, $V_k = \partial \theta_k / \partial \epsilon_k$ and we can obtain its updating equation by differentiating (7) with respect to ϵ_k :

$$V'_k = V_k + \sum_{i,j(i \neq j)} \left[e_{ji} \frac{\partial u_{ji}(\theta)}{\partial \theta_k} + \epsilon_k V_k \left(e_{ji} \frac{\partial^2 u_{ji}(\theta)}{\partial \theta_k^2} - \left(\frac{\partial u_{ji}(\theta)}{\partial \theta_k} \right)^2 \right) \right], \quad (37)$$

$k = 1, \dots, r$, where V'_k denotes the next time step's value of V_k ⁶. The set of (7), (36) and (37) forms the adaptive algorithm.

5 Experimental Results

In this chapter, we provide the experimental results to evaluate the algorithms proposed in the preceding chapter 4.

5.1 Setting of Simulation

First, we explain the formation of agents. We simulated 100 agents, including 70 good agents ($\bar{\theta}_k = 0.9$) and 30 bad ones ($\bar{\theta}_k = 0.1$). The ratings R_{ji} by good agents are generated by

$$R_{ji} = \prod_{[0,1]} [\bar{\theta}_i + 0.1N(0,1)], \quad (38)$$

where $N(0,1)$ denotes a normally distributed stochastic variable with zero mean and unit variance. As for bad agents, there are two types. The rating R_{ji} by the first type (15 agents) is generated randomly (an uniform stochastic variable), whereas the rating R_{ji} by the second type (15 agents) is generated by

$$R_{ji} = \prod_{[0,1]} [1 - \bar{\theta}_i + 0.1N(0,1)]. \quad (39)$$

That is, agents of the second type are lying agents whose ratings are always complementary.

Furthermore, we assumed the asynchronous situation where R_{ji} were obtained asynchronously. At each time step, we randomly selected two agents, and they rated each other and reported their ratings to the online reputation mechanism. We always used asynchronous algorithms in our simulation.

We set the parameters $\theta_{k,0} = 0.7$ ($\forall k$), $\epsilon_{k,0} = 0.05$ ($\forall k$), $\mu = 0.01$, $\epsilon_- = 0$, and $\epsilon_+ = 0.3$. We can't set $\theta_{k,0} = 0$ ($\forall k$) and $\theta_{k,0} = 0.5$ ($\forall k$), because these are fixed points of Joint and Lie algorithms, respectively.

⁶The initial value of V_k ($V_{k,0}$) is 0.

5.2 Performance Comparison of Algorithms

Here we provide the experimental results. Figure 3 shows the changes in the squared error of 7 algorithms, that is Weighted, LLS, Joint, Lie, LLS^w (LLS_w), Joint^w (Joint_w) and Lie^w (Lie_w) (all algorithms are of the asynchronous type). The squared error is defined as

$$\sum_k (\theta_k - \bar{\theta}_k)^2, \quad (40)$$

and each path of the squared errors represents the average of 30 simulations.

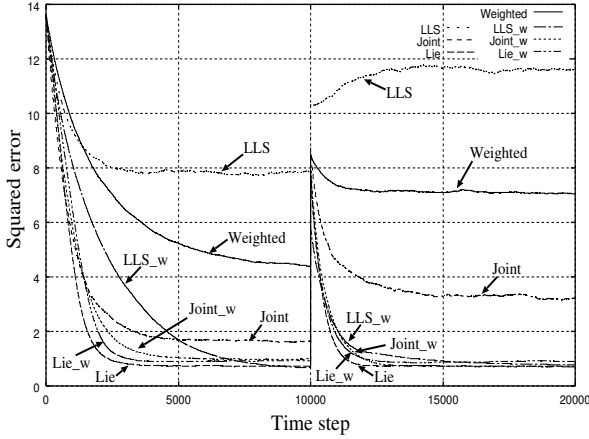


Figure 3: Changes in squared error.

Until time step 10,000, we can see that all algorithms, except for the LLS and Weighted algorithms, learn the true trustworthiness values $\bar{\theta}$ gradually and converge within the bounds of $[0, 2]$. The Weighted algorithm is more effective than the simplest LLS algorithm. However, LLS^w is much more effective than the Weighted algorithm, as we noted in 4.1.3.

Continuously, at the time step 10,000, we changed the true trustworthiness values θ_k of 10 good agents from 0.9 to 0.1, and these 10 agents started to express complementary ratings, i.e. (39). It is clear that again the LLS and Weighted algorithms do not work well and indeed make things worse. Moreover, this time we can see that the Joint algorithm cannot trace this change appropriately, although other algorithms, Lie, LLS^w, Joint^w and Lie^w, can adapt to this change immediately.

5.3 Effectiveness of Adaptive Algorithm

Here we present more results to show the effectiveness of the adaptive algorithm (see 4.3). Figure 4 shows the changes in squared error of 6 algorithms under the same condition in 5.2. Joint, Lie and LLS^w are exactly the same as in Figure 3, and Joint(a), Lie(a) and LLS_w(a) represent the corresponding algorithms with the adaptive algorithm.

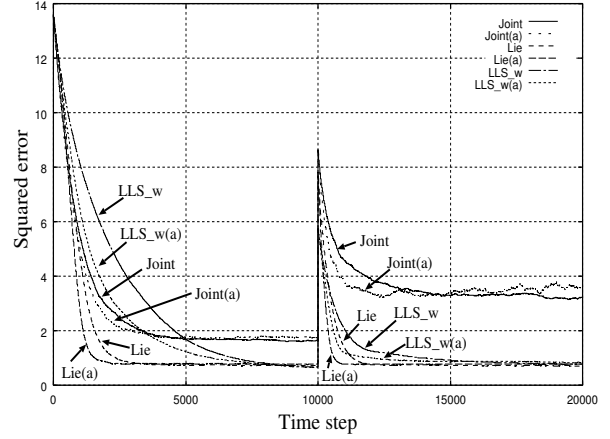


Figure 4: Changes in squared error (adaptive).

We can see that the adaptive algorithm improves the convergence rate of all three algorithms, not only in time steps before 10,000, but also after 10,000. We omitted the case of other algorithms, LLS, Joint^w and Lie^w, for the sake of a clear graph. However, improvements in the convergence rate by the adaptive algorithm are equally observed.

5.4 Discussion

In our simulation, all agents including malicious agents, have the static strategy of ratings. If the majority of agents are good, we consider that our proposed algorithms still work well under the situation containing malicious agents who dynamically change their rating strategies. However, empirical analysis is needed to examine the robustness against attacks by malicious agents with more intelligent manipulation strategies. Furthermore, game theory-based analysis will also help this robustness analysis. We may characterize our problem of online reputation mechanism as the nonzero-sum (general-sum) game between the online reputation mechanism and the intelligent malicious agents. This is future work.

6 Conclusion

We proposed stochastic approximation-based algorithms on our framework of the online reputation mechanism. These algorithms are obtained in a similar fashion to the corresponding rating models and cost functions. We extended these algorithms by introducing the adaptive algorithm of the step-size parameter, in addition to deriving asynchronous algorithms.

Experimental results show that the proposed algorithms can identify good and bad agents effectively under the conditions of the disturbances and also trace the changes in agents' true trustworthiness values adaptively.

References

- Albert Benveniste, Michel Métivier, and Pierre Priouret. *Adaptive Algorithms and Stochastic Approximations*. Springer-Verlag, 1990.
- Chrysanthos Dellarocas. Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC'00)*, 2000.
- Chrysanthos Dellarocas. The digitization of word-of-mouth: Promise and challenges of online reputation systems. In *Management Science*, 2003.
- Yoshiteru Ishida. An immune network approach to sensor-based diagnosis by self-organization. *Complex Systems*, 10:73–90, 1996.
- Peter Kollock. The production of trust in online markets. In E. J. Lawler, M. Macy, S. Thyne, and H. A. Walker, editors, *Advances in Group Processes (Vol. 16)*. Greenwich, CT: JAI Press, 1999.
- Harold J. Kushner and G. George Yin. *Stochastic Approximation and Recursive Algorithms and Applications (Second Edition)*. Springer-Verlag, 2003.
- Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. The MIT Press, 1998.
- Yao Wang and Julita Vassileva. Bayesian network-based trust model. In *Proceedings of the IEEE/WIC International Conference on Web Intelligence (WI'03)*, 2003.
- Bin Yu and Munindar P. Singh. Detecting deception in reputation management. In *Proceedings of Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'03)*, 2003.
- Giorgos Zacharia, Alexandros Moukas, and Pattie Maes. Collaborative reputation mechanisms in electronic marketplaces. In *Proceedings of the 32nd Hawaii International Conference on System Sciences (HICSS-32)*, 1999.

Multi-Agent Coordination in Tree Structured Multi-Stage Games

Katja Verbeeck

*Vrije Universiteit Brussel (COMO)
Pleinlaan 2 1050 Brussel, Belgium
kaverbee@vub.ac.be

Ann Nowé

†Vrije Universiteit Brussel (COMO)
Pleinlaan 2 1050 Brussel, Belgium
asnove@info.vub.ac.be

Maarten Peeters

‡Vrije Universiteit Brussel (COMO)
Pleinlaan 2 1050 Brussel, Belgium
mjpeeter@vub.ac.be

Abstract

A major problem in Multi-agent reinforcement learning research (MARL) is to let multiple agents learn how to coordinate to some equilibrium. Coordination in single stage problems, which are easily modeled as normal form games from game theory, is already studied profoundly. However, real-world problems are more naturally translated in multiple-stage problems. Multiple-stage problems can be modeled as Markov games, but learning in these models is not trivial. In this paper we introduce the notion of tree-model multi-stage games. An existing learning technique, called exploring selfish reinforcement learning (ESRL), which is based on learning automata theory and suited for coordination of pure independent agents, is extended to multi-stage games by using a hierarchy of automata. Experiments show that hierarchical exploring selfish reinforcement learning (HESRL) enables independent agents to coordinate in multi-stage tree structured games, of which the dual single stage game would be very large. Examples of multi-stage games include situations in which agents in a first stage have to decide with whom to cooperate; the sequel of the game is influenced by this decision.

1 Introduction

Coordination is an important issue in multi-agent reinforcement learning research, because it is often a requisite when agents want to maximize their revenue. In many real-world applications the problem of coordination becomes even harder because of system limitations such as, partial or non observability, communication costs, asynchronism etc. For instance in systems where resources are limited as in job scheduling and routing these assumptions certainly apply. Predefined rules are not feasible in complex and changing environments, even more communication has its price. Therefore we are interested in how independent reinforcement learning agents can learn how to coordinate. More specifically in this paper we are interested in whether independent agents are able to coordinate in sequential coordination problems.

As opposed to joint action learners, independent agents only get information about their own action choice and pay-off. As such, they neglect the presence of the other agents. Joint action learners, (Boutilier, 1996; Hu and Wellman, 1998; Chalkiadakis and Boutilier, 2003) do perceive the actions of the other agents in the environment and are therefore able to maintain models on the strategy of others. However in the light of the applications we have in mind the assumptions joint action learners make are too strong. We assume here that observations are not reliable, the environment is unknown to the agents and only limited communication is allowed.

In single stage games independent reinforcement learning agents are sometimes able to find optimal solutions, (M.Peeters, 2003), however very often some form of limited communication is required. We call such agents,

pseudo independent. Different techniques for (pseudo) independent agents exist, and guarantee them to find a global optimal behavior in single stage common interest games, see e.g. S.Kapetanakis et al. (2003); Verbeeck et al. (2003); J.Parent et al. (2004); M.Lauer and M.Riedmiller (2000).

Single stage games are useful testbeds for some real world problems such as job scheduling (Nowé et al., 2001; J.Parent et al., 2004), however in most real world problems a sequence of decisions has to be learned. In this paper we take the first step in scaling up our technique of exploring selfish reinforcement learning (ESRL), (Verbeeck et al., 2003; J.Parent et al., 2004) to multiple stage problems. For now we restrict ourselves to multi-stage common interest games for which the corresponding state graph is a tree, i.e. it shows no loops and has disjunct paths. But we also assume that rewards can be stochastic.

In ESRL, independent RL agents explore as many joint actions as possible. They do so by excluding actions from their private action space, so that the joint action space shrinks more and more. In combination with random restarts the algorithm is proved to converge in the long run to the optimal joint action of a single stage common interest game, (Verbeeck et al., 2003).

ESRL agents are based on the theory of learning automata, more in particular learning automata games, (Narendra and Thathachar, 1989). Modeling independent agents as learning automata is easily motivated. Learning automata are updated strictly on the basis of the response of the environment and not on the basis of any knowledge regarding other automata or their strategies. Moreover their behavior is already studied thoroughly both in a single automata setup as in interconnected systems, see

Narendra and Thathachar (1989); Thathachar and Sastry (2002).

We adapt the ESRL agents to hierarchical ESRL agents (called HESRL). Every agent in the system will now consists of several learners (learning automata in our case). More in particular an HESRL agent has a new learner for every state or node in the multi-stage tree. The HESRL agents are based on the hierarchical learning automata approach of (K.S.Narendra and Parthasarathy, 1988) which was used for hierarchical multi-objective analysis. We enhance them with the exploration and exclusion abilities of the ESRL agents, so that they will be able to converge to an optimal path in the multi-stage common interest tree.

In the next section we start with examples of tree structured multi-stage games. We compare the coordination problems of multi-stage common interest trees with those of single stage common interest games. Section 3 reviews learning automata basics and introduces the hierarchical framework of K.S.Narendra and Parthasarathy (1988). It is shown how the reward function of the multi-stage tree can be translated to the environment responses of the hierarchical learning automata. In section 4 the exclusion technique of ESRL agents is reviewed and extended so that HESRL agents can exclude paths from the tree in the same way ESRL agents exclude joint actions from the action space. Two possible ways of exclusion are considered, i.e. a breadth first and a depth first exclusion technique. Section 5 reports on the first results obtained for the multi-stage trees considered in section 2. We briefly conclude in the last section.

2 Tree Structured Multi-Stage Games

The foundation of single agent reinforcement learning in multiple state environments are formed by Markov decision processes, MDP's. In (Boutilier, 1996, 1999) this framework was extended for use in a multi-agent setting to MMDP's, i.e. multi-agent Markov decision processes. MMDP's are a form of Stochastic games (also called Markov games), (Hu and Wellman, 1998; Littman, 1994), but in an MMDP agents are fully cooperative¹ and therefore there is only a single reward function for all agents. Formally an MMDP is a tuple $\langle S, \alpha, \langle A_i \rangle_{i \in \alpha}, Pr, R \rangle$ where S is a set of states, α a set of agents, A_i a finite set of actions available to agent i , $Pr : S \times A_1 \times \dots \times A_n \times S \rightarrow [0, 1]$ a transition function and $R : S \rightarrow \mathbb{R}$ a reward function.

Figure 1 gives an example of a simple MMDP with two stages. It can be viewed as a standard MDP in which the actions are implemented in a distributed fashion. Similar to an MDP a credit assignment problem arises, however their is an added coordination problem, which makes it far more complicated. In the above problem the first agent

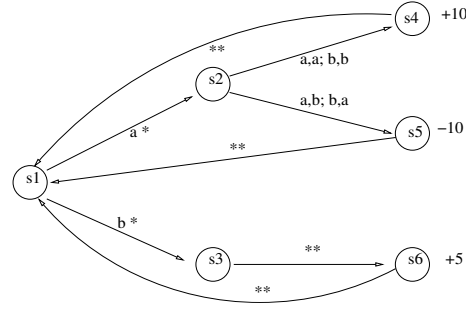


Figure 1: The Opt-In Opt-out game: A simple MMDP with a coordination problem and two stages from (Boutilier, 1996). Both agents have two actions, a or b . A^* can be either action a or b .

	aa	ab	ba	bb
aa	10	-10	10	-10
ab	-10	10	-10	+10
ba	5	5	5	5
bb	5	5	5	5

Figure 2: Translation of the multi-stage tree of figure 1 in the dual single stage common payoff game. A path in the multi-stage tree becomes a joint action in the single stage game.

should decide to play a in the first stage, no matter what the other agent decides and both agents have to coordinate their actions in the second stage to reach the high payoff state s_4 .

The tree of Figure 1 can also be viewed as a single stage game. Every sequence of two actions, an agent can take can be considered as a single action of that agent's action space. A path in the tree will be a joint action in the single stage game. In Figure 2 the corresponding game matrix is given. The optimal joint actions or Nash equilibria are (aa, aa) , (aa, ba) , (ab, ab) and (ab, bb) . The Nash equilibria of the dual single stage game are called equilibrium paths in the tree.

In (S.Kapetanakis et al., 2003) a number of coordination problems for single stage identical payoff games are analyzed. The climbing game, the penalty game and their descendants are generally accepted as hard coordination problems. The climbing game and the penalty game are given in respectively Figure 3 and Figure 4.

	a	b	c
a	11	-30	0
b	-30	7	6
c	0	0	5

Figure 3: The climbing game: A hard coordination problem.(from Claus and Boutilier (1998))

¹Each agent gets the same payoff.

	a	b	c
a	10	0	k
b	0	2	0
c	k	0	10

Figure 4: The penalty game: A hard coordination problem.(from Claus and Boutilier (1998))

Both problems are difficult to solve from the viewpoint of agent coordination. In the first game the punishment for mis-coordination in the neighborhood of the optimal joint action (a, a) is extremely high, and therefore convergence to it is very difficult for agents using only limited communication. In the penalty game 2 different optimal joint actions co-exist, however when the agents each choose to play the other optimal action, this mis-coordination is punished by a penalty term k . In both games the agents may be tempted to play the safe, non-optimal actions, which is (c, c) for the climbing game and (b, b) for the penalty game.

As can be expected the same coordination problems occur in sequential coordination problems. Figure 2 shows that punishments surround the optimal joint actions (i.e. (aa, aa) , (aa, ba) , (ab, ab) and (ab, bb)) as in the climbing game and different optimal joint action exists as in the penalty game. Again safe non-optimal joint actions exists, i.e. the first player can play either ba or bb , no matter what the second agent plays.

In sequential coordination problems these typical coordination problems can occur on every stage of the game. Consider for instance the multi-stage tree of Figure 5 which is a variant of the tree in Figure 1. In this case a coordination issue also applies in the first stage of the game, i.e. as well as in the first stage as in the second stage both agents should agree on the same action. So again 4 equilibrium paths exists which lead to state $s4$. Non coordination in the first stage is less worse than non-coordination in the second stage. The corresponding single stage game is given in Figure 6.

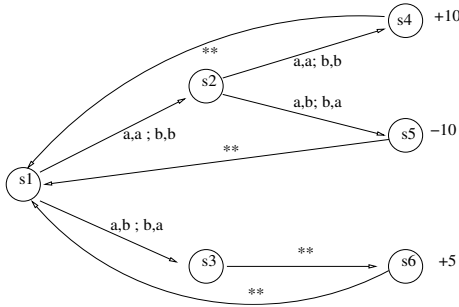


Figure 5: A simple MMDP with a coordination problem in both stages.

Another issue related to sequential coordination arises when a reward is also given after the first stage, instead of only a delayed reward at the end of the path. Just as

	aa	ab	ba	bb
aa	10	-10	5	5
ab	-10	10	5	5
ba	5	5	10	-10
bb	5	5	-10	10

Figure 6: Translation of the multi-stage tree of figure 5 in the dual single stage common payoff game.

in an MDP, discounting rewards becomes important. A path of which the first link gives bad reward, may belong to a path which receives high reward at the end. What will be the equilibrium path depends on how important immediate reward is opposed to for instance global or average reward on the whole path. Usually a discount factor $\gamma \in [0, 1]$ weights the importance of the immediate reward. γ should be chosen in function of the goal, cfr. dynamic programming techniques and reinforcement learning algorithms.

In Boutilier (1996) an MMDP is decomposed into local (single stage) state games and the agents search coordinated joint actions at the individual state games instead of trying to find a coordinated global policy. It is assumed that every agent knows the structure of the game and therefore is able to compute the optimal value function for the joint MDP. In Boutilier (1999) the agents reason explicitly about specific coordination mechanisms. An extension of the value iteration algorithm is studied in which the system's state space is augmented with the state of the coordination mechanism adopted. In our view the MMDP is a sequence of one stage games as in Boutilier (1996). The only assumption we make for now, is that the MMDP has a tree structure, i.e. there are no loops in the state diagram of the MMDP and paths are disjunct. Unlike Boutilier (1996) we allow that the rewards of the stage games are stochastic.

Exploring selfish reinforcement learning is able to overcome typical coordination problems in single stage games, (Verbeeck et al., 2003) even in case of stochastic rewards. In section 4 we enhance hierarchical agents with the same exploration abilities as ESRL agents have, so that independent HESRL agents are able to find optimal paths in MMDP trees. The next section first introduces hierarchical learning automata.

3 Hierarchical Learning Automata

In this section the theory of learning automata and automata games is briefly reviewed and hierarchical Learning automata are discussed. The latter will form the basis of the HESRL agents, which are introduced in the next section.

3.1 Learning Automata

A learning automaton formalizes a general stochastic systems in terms of states, actions, state or action probabilities and environment responses, see Narendra and Thathachar (1989); Thathachar and Sastry (2002). A learning automaton is a precursor of a policy iteration type of reinforcement learning algorithm and has some roots in psychology and operations research. The design objective of an automaton is to guide the action selection at any stage by past actions and environment responses, so that some overall performance function is improved. At each stage the automaton chooses a specific action from its finite action set and the environment provides a random response, see Figure 7.

In a variable structure stochastic automaton the probabilities of the various actions are updated on the basis of the information the environment provides. Action probabilities are updated at every stage using a reinforcement scheme. It is defined by a quadruple $\{\alpha, \beta, p, T\}$ for which α is the action or output set $\{\alpha_1, \alpha_2, \dots, \alpha_r\}$ of the automaton, β is a random variable in the interval $[0, 1]$, p is the action probability vector of the automaton or agent and T denotes an update scheme. The output α of the automaton is actually the input to the environment. The input β of the automaton is the output of the environment, which is modeled through penalty probabilities c_i with $c_i = P[\beta | \alpha_i], i : 1 \dots r$.

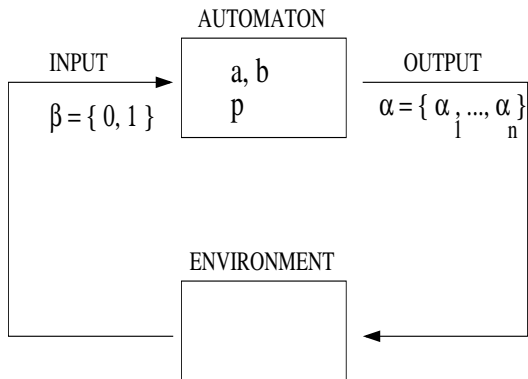


Figure 7: A Learning Automata - Environment pair

A linear update scheme that behaves well in a wide range of settings² is the linear reward-inaction scheme, denoted by (L_{R-I}) . The philosophy of this scheme is essentially to increase the probability of an action when it results in a success and to ignore it when the response is a failure. The update scheme is given by:

$$p_i(n+1) = p_i(n) + a(1 - \beta(n))(1 - p_i(n))$$

if α_i is chosen at time n

² (L_{R-I}) is what is called absolutely expedient and ϵ optimal in all stationary random environments. This means respectively that the expected average penalty for a given action probability is strictly monotonically decreasing with n and that the expected average penalty can be brought arbitrarily close to its minimum value.

$$p_j(n+1) = p_j(n) - a(1 - \beta(n))p_j(n)$$

if $\alpha_j \neq \alpha_i$

The constant a is called the reward or step size parameter and belongs to the interval $[0, 1]$. In stationary environments $p(n)_{n>0}$ is a discrete-time homogeneous Markov process and convergence results for (L_{R-I}) are obtained. Despite the fact that multiple automata environments are non-stationary, the (L_{R-I}) scheme is still appropriate in learning automata games.

3.2 Learning Automata Games

Automata games were introduced to see if automata could be interconnected in useful ways so as to exhibit group behavior that is attractive for either modeling or controlling complex systems. A play $\alpha(t) = (\alpha^1(t) \dots \alpha^n(t))$ of n automata is a set of strategies chosen by the automata at stage t . Correspondingly the outcome is now a vector $\beta(t) = (\beta^1(t) \dots \beta^n(t))$. At every instance all automata update their probability distributions based on the responses of the environment. Each automaton participating in the game operates without information concerning payoff, the number of participants, their strategies or actions.

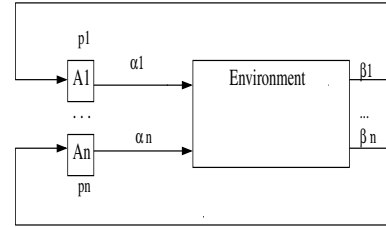


Figure 8: Automata Game formulation.

The following results were tested and proved in Narendra and Thathachar (1989): In zero-sum games the (L_{R-I}) scheme converges to the equilibrium point if it exist in pure strategies, i.e. if there is a pure Nash equilibrium. In identical payoff games as well as some non-zero-sum games it is shown that when the automata use a (L_{R-I}) scheme the overall performance improves monotonically. Moreover if the identical payoff game is such that a unique equilibrium point exists, convergence is guaranteed. In cases where the game matrix has more than one equilibria the (L_{R-I}) scheme will converge to one of the Nash equilibria. The solution obtained depends on the initial conditions.

Learning automata games form the basis for ESRL agents. In the same way hierarchical learning automata form the basis for HESRL agents.

3.3 Hierarchical Learning Automata

For our approach, we were inspired by the work of K.S.Narendra and Parthasarathy (1988) in which several

hierarchies of learning automata are able to solve a sequence of stochastic identical payoff games at various levels.

A simple hierarchical system of learning automata as in figure 11 can be thought of as a single automaton (or agent) whose actions are the union of actions of all automata at the bottom level of the hierarchy. We call this agent a hierarchical agent. When different step sizes are used at the various levels, the single automaton can still be absolutely expedient³, (K.S.Narendra and Parthasarathy, 1988). The hierarchical agent can be generalized further to situations where each automaton acting at any level receives a response from a local environment in addition to the the global response obtained at the end of the cycle. An example is given in figure 12. In this example two hierarchical agents interact at two different levels. They receive an environment response on each level. As such they will be able to learn and play the two stage trees given in section 2.

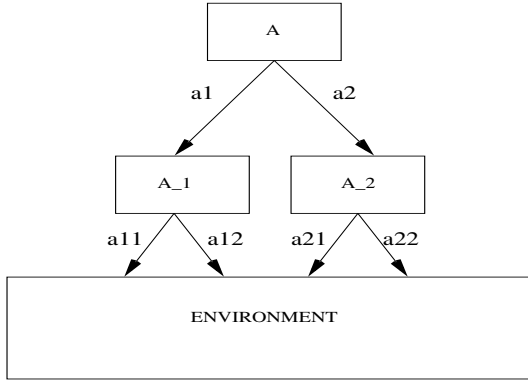


Figure 9: A simple hierarchical system of learning automata with two stages, from K.S.Narendra and Parthasarathy (1988)

The interaction of the two hierarchical agents of figure 12 goes as follows. At the top level (or in the first stage) agent 1 and agent 2 meet each other in the stochastic game M . They both take an action using their top level learning automata A and B . Performing actions a_i by A and b_j by B is equivalent to choosing automaton A_i and B_j to take actions at the next level. The response of environment $E1$, $\beta_1 \in \{0, 1\}$, is a success or failure, where the probability of success is given by m_{ij} . At the second level the learning automata A_i and B_j choose their actions a_{ik} and b_{jl} respectively and these will elicit a response β_2 from environment $E2$ of which the probability of getting a positive reward is given by $m'_{ik,jl}$. At the end all the automata which were involved in the games, update their action selection probabilities based on the actions performed and the response of the composite environment, i.e. $\beta(n) = \lambda\beta_1(n) + (1 - \lambda)\beta_2(n)$, where

³This means that the expected average penalty for a given action probability is strictly monotonically decreasing with time.

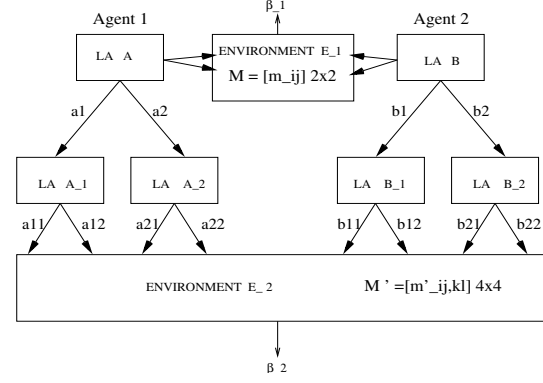


Figure 10: Interaction of two hierarchies of learning automata at two stages, from K.S.Narendra and Parthasarathy (1988)

$\lambda \in [0, 1]$.

To let these agents play for instance the tree game of figure 1, we have to map the immediate rewards in the tree to the environment responses of the hierarchical learning automata system. We translate the rewards of figure 1 in stochastic rewards, i.e. we scale the rewards of the tree in figure 1 between 0 and 1 and use these results as a probability of success⁴ in the update of the learning automata. In figure 1 no rewards are given after the first stage, so M becomes the null matrix as is shown in figure 11. After the second stage possible rewards are 10, -10 and 5, which gives scaled probabilities of 1, 0 and 0.75 for M' respectively. The actions taken by the agents in the first stage are not neglectable, as they determine the game of the next stage and the learning automata which are going to play it. In the tree of figure 1 all rewards are given at the end, therefore we should set the weight factor $\lambda = 0$.

In general the hierarchical agent should have as many levels of automata as there are stages in the tree and a suitable weight factor λ (cfr a discount factor in RL). The matrices are not known to the agents, i.e. learning is model-free.

When every automaton of the hierarchy uses a linear reward inaction scheme and the step sizes of the lower levels automata vary with time, the overall performance of the hierarchical learning automata system will improve at each stage, (K.S.Narendra and Parthasarathy, 1988). At any stage n the step size of all the lower level automata taking part in the game is changed to:

$$a(n) = \frac{a(n-1)}{p_i(n)}$$

where i is the action taken by this automaton during the previous game iteration, and $p_i(n)$ is the probability this

⁴The reason for this is that we use what is called a P-model learning automata. This means that the reward they receive from their environment is binary, i.e. the action was a success or a failure. This is however no limitation, as other richer learning automata exist for which the P-model algorithms are extended.

$$M = \begin{pmatrix} 0.0 & 0.0 \\ 0.0 & 0.0 \end{pmatrix}$$

$$M'(0,0) = \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix}$$

$$M'(0,1) = \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix}$$

$$M'(1,0) = \begin{pmatrix} 0.75 & 0.75 \\ 0.75 & 0.75 \end{pmatrix}$$

$$M'(1,1) = \begin{pmatrix} 0.75 & 0.75 \\ 0.75 & 0.75 \end{pmatrix}$$

Figure 11: Translation of the multi-stage tree of figure 1 in environment responses for the hierarchical learning automata. The joint action chosen at the first stage, decides the game that is played at the second stage.

automaton has for action i after iteration n . However in the experiments we will see that hierarchical agents will not always converge to optimal paths. Therefore we turn to hierarchical exploring selfish reinforcement learning.

4 Hierarchical Exploring Selfish Reinforcement Learning

4.1 ESRL

The technique of Exploring Selfish Reinforcement Learning was introduced in Nowé et al. (2001) and Verbeeck et al. (2003) for respectively single stage games of conflicting interest and single stage games of common interest⁵. We focus on the common interest version of the algorithm here. The main idea of the technique is to explore the joint action space $\langle A_i \rangle_{i \in \alpha}$ of the agents as efficient as possible by shrinking it temporarily during different periods of play. At the beginning of learning, agents behave selfish or naive; i.e. they ignore the other agents in the environment and use a (L_{R-I}) reward-inaction learning automata update scheme to maximize their payoff. The first period of play ends when the agents have found a Nash equilibrium. As mentioned in section 3.2 an (L_{R-I}) scheme will converge to one of the Nash equilibria of the learning automata game, under a suitable step size. So the agents do converge after a certain number of iterations⁶.

⁵In these references we did not call our technique ESRL yet.

⁶In this paper the number of iterations done in one period (we call it the period length) is chosen in advance and thus a constant. In a newer version of the algorithm, the agents themselves learn when they are con-

```

## EXCLUSION PHASE

if (convergence_has_happened) {

    action := action_converged_to ;

    if (new_payoff(action) > best_payoff_so_far(action)) {
        best_payoff_so_far(action) := new_payoff(action) ;
    }

    if (more_than_2_actions(my_actionSet) ) {
        my_actionSet := my_actionSet - action;
    }
    else {
        my_actionSet := my_original_actionSet ;
    }

    random_initialize_prob_of_not_excl_actions;

}

```

Figure 12: Pseudo code of the exclusion phase for the common-interest ESRL agents.

Which Nash equilibrium the agents will find is not known in advance, it depends on the initial conditions and the basin of attraction of the different equilibria.

Next all the agents exclude the action they converged to in the previous period of play. If the average payoff for this action was better than the average payoff they received so far for that action, they store this average as a new best payoff so far for that action. We call this part of the algorithm the exclusion phase.

A new period of selfish play can now restart in a smaller joint action space and convergence to a new joint action will take place. As such agents alternate between playing selfish and excluding actions. When the agents have no actions left over in their individual action space, the original action space is restored and learning restarts in the full joint action space⁷. To enlarge the possibility that they shrink the joint action space in as many ways as possible, and thus collecting information about as many joint actions as possible⁸, they take random restarts, i.e. they initialize their action probabilities (of the not excluded actions) randomly at the start of a new period.

As agents remember the best payoff they received for each action, ESRL was proved to converge in fully cooperative games to the optimal solution without needing communication, even in stochastic environments Verbeeck et al. (2003). The pseudo-code of the exclusion part of the algorithm is given in figure 12.

So thanks to action exclusions and random restarts,

verged.

⁷At least for symmetric games in which all agents have the same number of actions.

⁸Actually this means that there should be enough exploration.

ESRL agents avoid to converge to local optima. We will enhance hierarchical agents with the same abilities to let them avoid to converge to a suboptimal path in the multi-stage tree.

4.2 HESRL

HESRL agents are hierarchical agents with added exploration abilities. The idea for HESRL agents is to let them converge to a path in the multi-stage tree and then exclude that path or a part of that path from the tree, so that in a new period of play other paths can be explored in a smaller joint path space, i.e. the space of all possible paths. So initially HESRL agents behave as common hierarchical agents from section 3.3; and after enough iterations they will converge to one path in particular, though not necessary the optimal one. As the actions for the hierarchical agents in multi-stage trees are sequences of local actions, the exclusion phase can now be defined in different ways.

A first approach could be, a *breadth first* method of excluding. This means that the agents exclude only the first link of the path. More concrete this means that the learning automata of the first stage of every agent excludes the action that it converged to. So all paths which start from that link will be excluded. After only a few periods of selfish play, all actions of the first level automaton will be excluded and thus all paths will be excluded. If this is the case all excluded actions of the first level automata of all the agents are freed again, and a new period of selfish play restarts in the original joint path space. Action probabilities of all learning automata are initialized randomly at the beginning of every period so that as many paths as possible can be found. The pseudo code of breadth first exclusions is given in figure 13.

Alternatively, an agent could randomly decide to exclude any action involved in the path. In a second approach the agents exclude the action from that learning automata, that was involved on the bottom level. We call this approach a *depth first* exclusion method. Of course after a while all the actions of an automaton on the bottom level could become excluded. In that case the action of the automaton of the previous level, which leads to the corresponding bottom level automaton should also be excluded. So after the first period of selfish play, exactly one path becomes excluded. After several periods of play more paths become excluded and when eventually exclusions reach the first level automaton of the agent, all the paths become excluded and thus the joint path space should be freed again. Again randomization on the action probabilities of all learning automata at the different levels is done so as to increase the number of different paths the agents converge to. The pseudo code of depth first exclusions is given in figure 14.

In the next section hierarchical agents and HESRL agents are tested against the multi stage trees of section 2. Furthermore, breadth-first and depth-first exclusions are

```

## EXCLUSION PHASE

if (convergence_has_happened) {

    path := path_converged_to;
    action := first_action(path);
    new_payoff := payoff(path);

    LA := sequence_of_LA_active_in_path;
    Top_LA := first_(LA);

    if (new_payoff > best_payoff_so_far(action) {
        best_payoff_so_far(action) := new_payoff ;
    }

    if (more_than_2_actions(actionSet_TopLA) {
        actionSet_TopLA := actionSet_TopLA - action;
    }
    else {
        actionSet_TopLA := original_actionSet_TopLA ;
    }

    for_each_LA DO {
        random_initialize_prob_of_not_excl_actions;
    }

}

```

Figure 13: Pseudo code of the exclusion phase for the common-interest HESRL agent with breadth-first exclusions.

evaluated and compared.

5 Experiments

In this section we report on the results of hierarchical agents and HESRL agents playing different multi-stage games. In a first subsection we test how well the agents can cope with the typical coordination problems mentioned in section 2. The next subsection studies the effect of discounted rewards. In the last subsection breadth first exclusions and depth first exclusions are compared. In all experiments the step-size parameter α is initially set to 0.05. Note that the experiments with the HESRL agents only show the exploration phase, i.e. after enough periods are played in which different joint actions are found an exploitation phase should be added in which the agents play the best path they found.

5.1 Hard Coordination problems

As seen in section 2, the multi-stage game of figure 1 combines two hard coordination problems. A first problem is that multiple equilibrium paths exists, so that agents have to coordinate on the same equilibrium. Secondly the equilibrium path is surrounded by low reward paths. In figure 15 the average payoff for hierarchical agents⁹ is given for the game of figure 1. The average is approximately 0.8. This means that in almost every run of the game the hierarchical agents converge to one of the suboptimal paths of the tree, which gives on average a payoff of 0.75. In only a few runs an optimal path, which gives payoff 1.0 is reached.

Figure 16 and figure 17 give the average of a typical run of HESRL agents using respectively breadth-first and depth-first exclusions. The period length is set to 1000 iterations, i.e. after 100 iterations of playing selfish, the agents run their exclusion phase of which the pseudo code is given in section 4. In both experiments an optimal path is reached. The breadth-first HESRL agent in figure 16 find an optimal path in the first and fourth period of selfish play. The depth-first HESRL agents find an optimal path in the third and fourth period. Over 20 different runs both types of HESRL agents find an optimal path in all runs.

The game tree of figure 5 has 2 coordination problems, one on both stages of the game. It appears that hierarchical agents perform slightly better on this problem. Figure 18 shows that hierarchical agents now reach an average of approximately 0.93. So they find on average optimal paths more frequently than for the previous problem. When we compare the dual single stage games of these multi-stage trees, given in respectively figure 2 and figure 6 we can see that this may not come as a surprise. Both games have the same level of difficulty. Although the second game has a coordination problem on every stage of the game, this doesn't make the overall single

```

## EXCLUSION PHASE

if (convergence_has_happened) {

  path := sequence_of_actions_converged_to;
  action := last_action(path);
  new_payoff := payoff(path);

  LA := sequence_of_LA_active_in_path;
  Top_LA := first(LA)
  Bottom_LA := last_LA(LA)

  if (new_payoff > best_payoff_so_far(action) {
    best_payoff_so_far(action) := new_payoff;
  }

  actionSet := actionSet(Bottom_LA);
  actionSet := actionSet - action;

  while (actionSet = emptySet) and
    (Bottom_LA ≠ Top_LA) {
    path := path - action;
    LA := LA - Bottom_LA;
    action := last_action(path);
    Bottom_LA := last_LA(LA);
    actionSet := actionSet(Bottom_LA);
    actionSet := actionSet - action;
  }

  if (Bottom_LA = Top_LA) and (actionSet = emptySet) {
    for_each_LA DO {
      actionSet(LA) := original_actionSet(LA);
    }
  }

  for_each_LA DO {
    random_initialize_prob_of_not_excl_actions;
  }
}

```

Figure 14: Pseudo code of the exclusion phase for the common-interest HESRL agent with depth-first exclusions.

⁹The average is calculated over 100 different runs.

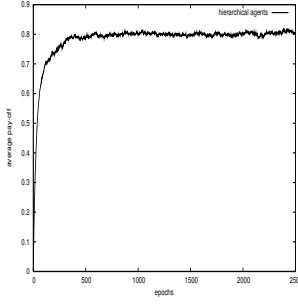


Figure 15: The average payoff for hierarchical agents playing the multi-stage tree of figure 1.

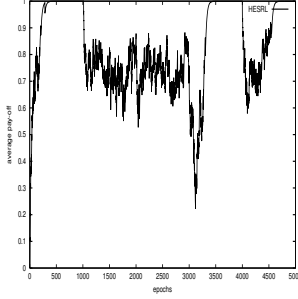


Figure 16: The average payoff for HESRL agents with breadth-first exclusions playing the multi-stage tree of figure 1.

stage game more difficult. In fact, the Nash equilibria are now situated on the diagonal, which apparently makes it easier for the hierarchical agents to reach them.

The results for HESRL agents are given in figure 19 and figure 20. Again both types find an optimal path. The same period length is used, i.e. 1000 iterations. Again a 100% converge to one of the equilibrium paths is reached when different runs are played.

In figure 19 you can see that after the first two periods, i.e. from epoch 3000 until 5000, the agents are reaching a suboptimal path, whereas in figure 20 the agents find an optimal path in the first 4 periods. This is because the agents use a different exclusion technique. Indeed, in figure 19 the agents have found 2 equilibrium paths,

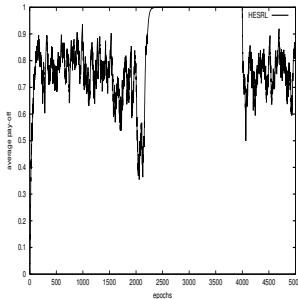


Figure 17: The average payoff for HESRL agents with depth-first exclusions playing the multi-stage tree of figure 1.

i.e. $(aa, **)$ and $(bb, **)$, but after these 2 periods, the agents have excluded all paths that begin with action aa and action bb , so only suboptimal paths are left over in the joint path space, while actually there 4 different joint equilibrium paths. With depth-first exclusions only one path is excluded from the joint path space at a time, so the depth-first HESRL agents can find all equilibrium paths in sequential periods, that is exactly what happens as is shown by figure 20.

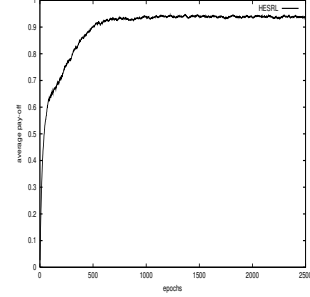


Figure 18: The average payoff for hierarchical agents playing the multi-stage tree of figure 5.

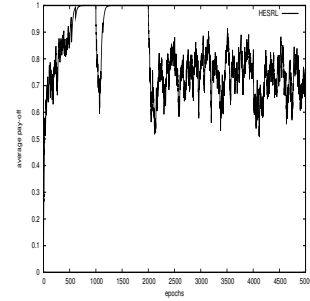


Figure 19: The average payoff for HESRL agents with breadth-first exclusions playing the multi-stage tree of figure 5.

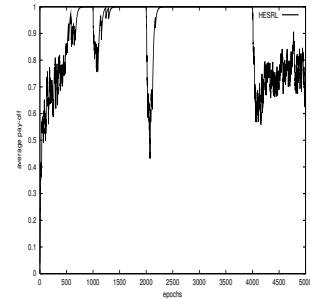


Figure 20: The average payoff for HESRL agents with depth-first exclusions playing the multi-stage tree of figure 5.

5.2 Discounting Rewards

Figure 23 gives an example of a sequential game problem which gives immediate reward after every stage. It

was already reported in (K.S.Narendra and Parthasarathy, 1988). We played it with hierarchical agents and HESRL agents using a weight factor of $\gamma = 0.5$. The dual single stage game is given in figure 22. Important to notice is that the equilibrium of this game is situated at the joint action (aa, bb) with an average reward of 0.725. So in the first stage agent 1 should choose action a and agent 2 should choose action b . However this is not the optimal joint action of the first stage game M , i.e. joint action (a, b) gives an average payoff of 0.6, while joint action (a, a) gives an average payoff of 0.7.

$$M = \begin{pmatrix} 0.7 & 0.6 \\ 0.1 & 0.1 \end{pmatrix}$$

$$M'(0, 0) = \begin{pmatrix} 0.6 & 0.2 \\ 0.3 & 0.1 \end{pmatrix}$$

$$M'(0, 1) = \begin{pmatrix} 0.3 & 0.85 \\ 0.2 & 0.2 \end{pmatrix}$$

$$M'(1, 0) = \begin{pmatrix} 0.4 & 0.1 \\ 0.2 & 0.1 \end{pmatrix}$$

$$M'(1, 1) = \begin{pmatrix} 0.3 & 0.2 \\ 0.2 & 0.2 \end{pmatrix}$$

Figure 21: A sequential stage game from K.S.Narendra and Parthasarathy (1988). The joint action chosen at the first stage, decides the game that is played at the second stage. $\lambda = 0.5$

Notice that there is only one optimal path in this game, the dual single stage game has a unique Nash equilibrium. This explains why hierarchical agents always converge to the optimal path in this game, as can be seen in figure 23. Their average payoff is approximately 0.725, which is exactly the Nash equilibrium payoff of the dual single stage game. From a coordination point of view, is this an easy coordination problem.

	aa	ab	ba	bb
aa	0.65	0.45	0.45	0.725
ab	0.5	0.4	0.4	0.4
ba	0.25	0.1	0.2	0.15
bb	0.15	0.1	0.15	0.15

Figure 22: The dual single stage game of the sequential game given in figure 21 with $\lambda = 0.5$.

This is also shown by the results of the HESRL agent, see figures 24 and 25. Convergence to the optimal path

happens always in the first period.

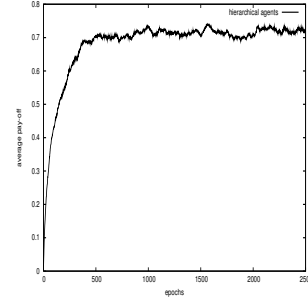


Figure 23: The average payoff for hierarchical agents playing the sequential game problem of figure 21.

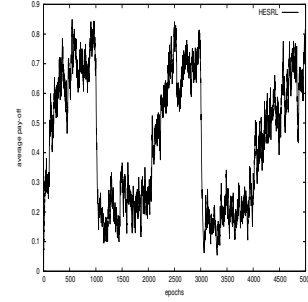


Figure 24: The average payoff for HESRL agents with breadth-first exclusions playing the sequential game problem of figure 21.

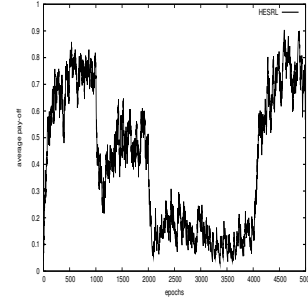


Figure 25: The average payoff for HESRL agents with depth first exclusions playing the sequential game problem of figure 21.

5.3 Large Joint Path Space

To put the HESRL exclusion techniques to the test and to make a comparison between depth-first and breadth-first exclusions we experimented with a larger joint path space. We scaled the game of figure 5 up to three players, all with three actions, i.e. a , b or c . This results in $3^3 = 27$ different paths possible. The equilibrium paths are reached when all players play the same action in every stage, for instance (abc, abc, abc) is an equilibrium path. In total there are 9 equilibrium paths. On average over 10 runs of

120.000 iterations the breadth-first technique managed to converge to an equilibrium path in 80% of the runs. This good result is due to the fact that the large joint path space is quickly reduced with this technique. And because of the many restarts the agent is still able to find an equilibrium path. We also ran this experiment with the depth-first technique. Here we only reached an equilibrium path 4 out of 10 times. The depth-first exclusion technique caused a slower reduction of the amount of paths resulting in a lower percentage of optimality.

6 Discussion

We made a first step in the direction of independent agents learning a sequence of actions in stochastic multi-stage environments with only limited communication. For this we use hierarchical exploring selfish RL agents, of which the main idea is to exclude actions locally and as such shrink the joint path space.

For now we only experimented with tree structured multi-stage games, however some interesting problems possess this structure. In J.Zhou et al. (1999) a two level tree structured set-up is studied in which the first level represents a group decision concerning the game environment to be played and the second level represents the choice of action within the selected environment. The issue under study is the effect of delays in the exchange of local information. It would be interesting to let independent HESRL agents play these games.

HESRL agents proved to be better than hierarchical LA agents in tree structured multi-stage games with hard coordination problems. Even games with multiple coordination problems at different stages of the game didn't appear more difficult for the HESRL agents. When rewards are discounted, HESRL agents still find an equilibrium path, even when suboptimal paths exist with a high reward on the first stage.

Two types HESRL agents were tested and compared, i.e. HESRL breadth-first excluding agents and HESRL depth-first excluding agents. The breadth-first excluding agents seem to be a better choice in larger joint path spaces. However more experiments should confirm this.

For now HESRL agents use a constant period length, however in the latest version of the ESRL algorithm, the agents decided themselves when they are converged to a Nash equilibrium. This usually results in shorter periods. The same behavior can be implemented for the HESRL agents.

We didn't investigate the effect of stochastic transitions yet, however we believe that stochastic transitions will have the same effect as playing an equivalent game with deterministic transitions but adapted stochastic payoffs. We believe that as long as the noise is reasonable low the HESRL agents will cope with it, however further investigations are necessary.

References

- C. Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the Sixth Conference on Theoretical Aspects of Rationality and Knowledge*, pages 195 – 210, Renesse, Holland, 1996.
- C. Boutilier. Sequential optimality and coordination in multiagent systems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 478 – 485, Stockholm, Sweden, 1999.
- G. Chalkiadakis and C. Boutilier. Coordination in multiagent reinforcement learning: A bayesian approach. In *Proceedings of the 2nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS-03)*, pages 709 – 716, Melbourne, Australia, 2003.
- C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the fifteenth National Conference on Artificial Intelligence*, pages 746 – 752, 1998.
- J. Hu and M.P. Wellman. Multi agent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 242–250, 1998.
- J.Parent, K.Verbeeck, A.Nowe, K.Steenhaut, J.Lemeire, and E.Dirckx. Adaptive load balancing of parallel applications with social reinforcement learning on heterogeneous systems. *Journal for Scientific Programming*, to appear, 2004.
- J.Zhou, E.A. Billard, and S. Lakshmivarahan. Learning in multilevel games with incomplete information-part ii. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 29(3):340–349, 1999.
- K.S.Narendra and K. Parthasarathy. Learning automata approach to hierarchical multiobjective analysis. Technical Report No. 8811, Electrical Engineering. Yale University., New Haven, Connecticut., 1988.
- M.L. Littman. Markov games as a framework for multiagent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157 – 163, 1994.
- M.Lauer and M.Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of the seventeenth International Conference on Machine Learning*, 2000.
- M.Peeters. A study of reinforcement learning techniques for cooperative multi-agent systems. Technical Report Masters thesis, Vrije Universiteit Brussel, Brussels, Belgium, 2003.
- K. Narendra and M. Thathachar. *Learning Automata: An Introduction*. Prentice-Hall International, Inc, 1989.

- A. Nowé, J. Parent, and K. Verbeeck. Social agents playing a periodical policy. In *Proceedings of the 12th European Conference on Machine Learning*, pages 382–393, Freiburg, Germany, 2001. Springer-Verlag LNAI2168.
- S.Kapetanakis, D. Kudenko, and M. Strens. Learning to coordinate using commitment sequences in cooperative multi-agent systems. In *Proceedings of the Third Symposium on Adaptive Agents and Multi-agent Systems, (AISB03) Society for the study of Artificial Intelligence and Simulation of Behaviour.*, 2003.
- M.A.L. Thatthachar and P.S. Sastry. Varieties of learning automata: An overview. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 32(6):711–722, 2002.
- K. Verbeeck, A. Nowé, and K. Tuyls. Coordinated exploration in stochastic common interest games. In *Proceedings of the Third Symposium on Adaptive Agents and Multi-agent Systems, (AISB03) Society for the study of Artificial Intelligence and Simulation of Behaviour.*, 2003.

A Role Based Model for Adaptive Agents

Danny Weyns*, Kurt Schelfthout*, Tom Holvoet* and Olivier Glorieux

*AgentWise, DistriNet

Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium

{Danny.Weyns, Kurt.schelfthout, Tom.Holvoet}@cs.kuleuven.ac.be

Abstract

This paper presents a model for adaptive agents. The model describes the behavior of an agent as a graph of roles, in short a *behavior graph*. Links between roles provide conditions that determinate whether the agent can switch roles. The behavior graph is assigned at design time, however adaptive role selection takes place at runtime.

Adaptivity is achieved through factors in the links of the behavior graph. A factor models a property of the agent or its perceived environment. When an agent can switch roles via different links, the factors determine the role the agent will switch to. By analyzing the effects of its performed actions the agent is able to adjust the values of specific factors, adapting the selection of roles in line with the changing circumstances.

Models for adaptive agents typically describe how an agent dynamically selects a behavior (or action) based on the calculation of a probability value as a function of the observed state for each *individual* behavior (or action). In contrast, the model we propose aims to dynamically adapt logical *relations* between different behaviors (called roles here) in order to dynamically form paths of behaviors (i.e. sequences of roles) that are suitable for the current state.

To verify the model we applied it to the Packet-World. In the paper we discuss simulation results that show how the model enables the agents in the Packet-World to adapt their behavior to changes in the environment.

1 Introduction

Adaptability is a system's capacity to take into account unexpected situations. Multi-agent systems are particularly characterized by the property that not everything can be anticipated in advance. In the context of cognitive agent systems the problem of adaptation is tackled by introducing learning techniques. Traditional learning techniques however do not fit the approach of behavior based agents since these agents do not build a symbolic model of their environment.

To deal with the problem of adaptation an agent has to take into account the quality of the effects realized by its past decisions. Several techniques for behavior based agent architectures have been proposed to realize adaptation, some examples are Maes and Brooks (1991), Drogoul (1993) or Bonabeau et al. (1998). In our research group we also developed two architectures for adaptive agents, see Schelfthout and Holvoet (2002) and Wolf and Holvoet (2003). All these models typically describe how an agent dynamically selects a behavior (or action) based on the calculation of a probability value in function of the observed state for each *individual* behavior (or action). The contribution of this paper is a model that aims to dynamically adapt logical *relations* between different behaviors (called roles here) in order to dynamically form paths of behaviors (i.e. sequences of roles) that are suitable for the current state.

This paper is structured as follows. Section 2 introduces the basic model for agent behavior. In section 3 we extend the model for adaptive behavior. Section 4 evaluates the model for the Packet-World application. We show how the model enables the agents in the Packet-World to adapt their behavior to changes in the environment. Finally we conclude in section 5.

2 Basic model for agent behavior

In this section we introduce the basic model for agent behavior. The basic model describes the behavior of an agent as a graph, in short a *behavior graph*. A behavior graph consists of *roles* that are connected by means of *links*. An agent executing a certain behavior graph is "moving" through this graph, executing actions in a role and moving to the next role via links connecting the roles. We now describe these concepts in detail.

2.1 Roles

A role is an abstract representation of a sequence or combination of actions the agent executes when it is in that role. A role abstracts from the number and detail of its actions as well as the applied action selection mechanism within the role. At each moment in the agent's lifetime there is exactly one active role, called the *current role*, all

other roles are inactive.

We choose the term role instead of task for the reason that a role is an organizational concept: it captures the fact that an agent is part of a multi-agent system, and thus part of an organization. We will also introduce the concept of *dependency roles* later to explicitly model both inter-agent dependencies, as well as a form of intra-agent parallelism.

The decision which combination of actions is grouped into a role is left to the human designer. He or she can choose to use atomic actions as a role: this leads to agents consisting of many fine-grained roles, and thus many links between those roles. It is likely that such a design generates a lot of overhead when executed. On the other hand, the designer can choose to group many actions into a few big roles. This makes the behavior graph look very comprehensive, while hiding a lot of complexity in the roles themselves. In practice the designer likely should strike a golden mean, taking into account the tradeoff between the overhead of minimal roles and the complexity of weighty roles.

2.2 Links

Besides roles, the behavior graph consists of directed links that connect roles. A link starts at a *source role* and points to a *goal role*. Several links can start from the same role and several links can point to the same role. When an agent is executing a certain role, it can decide that it wants to switch roles to any one role that is linked with the current role.

Additionally, a *condition* is associated with each link, that determines whether the link is *open* or *closed*. The condition is evaluated at runtime, and influences the selection of the next role the agent can execute: if a condition closes a link, the agent cannot switch to that link's goal role. An agent can only switch roles via open links.

These conditions are used to model that during execution, certain goal roles can not be executed (or it makes no sense executing them) from certain source roles. Links represent a logical connection between roles, based on the current context of the agent. Some roles are independent of each other - these are not connected at all, or only through intermediate roles. The links in the behavior graph represent all possible paths the agent can follow through the graph in order to reach its goals.

2.3 Role pre-conditions

The condition of a link determines whether roles can be switched via that link, i.e. whether the goal role can become current role. Often however, a role requires certain general conditions before it can be activated. For example a role may require particular resources in the environment to become active. We call such general conditions the *pre-conditions* of that role.

It is possible to integrate pre-conditions as conditions in each link that points to the role. However, this would require a copy of the pre-condition in each link. Therefore we decided to integrate pre-conditions in the roles themselves. Putting pre-conditions in roles promotes reuse of the role since it uncouples the links from the context of the roles (since the latter can be integrated in the pre-condition).

In summary, a switch from role A to role B is only possible if

- role A is the current role
- there is a link L from A to B;
- link L's condition is true, and L is thus open;
- role B's pre-condition is true.

2.4 Dependency roles

It is obvious that in general an agent must be able to perform identical actions in several different roles, as well as be able to respond to events that occur asynchronously and in parallel to the agent's current activities. An example is an agent that is able to respond to requests for information. When the agent receives a request it should be able to answer, no matter in which particular role the agent is at that moment. If we wanted to model this in the approach described above, we would have to duplicate the answering functionality in each role.

To avoid this duplication a new role can be introduced that is responsible to handle requests. All roles from which the agent must be able to answer requests are connected to this new role. Because requesting agents *depend* on the answers produced by this new role we call such role a *dependency role*. Dependency roles are similar to roles in the dependencies layer of the Cassiopeia approach, described in Drogoul and Zucker (1998).

Fig. 1 depicts a simple example of a behavior graph. In this example, the agent has two roles, *Move* and *Work*. An agent without work todo is in the *Move* role. In this role the agent moves around looking for work. When it detects work, the *Work Detected* link opens and the agent switches to the *Work* role. As soon as the work is finished the agent again switches to the *Move* role via the *Work Finished* link. The right part of Fig. 1 illustrates the common dependency of both roles to answer to a request of the headquarters to send the agents' current location. This functionality is modelled as the *Send Position* dependency role on top of the behavior graph.

Whereas a role of the behavior graph becomes active through role switching via a link, a dependency role becomes active via the connection with the current role when a dependency is detected. The dependency role then gets priority over the current role to resolve the dependency after which the underlying role takes over again. If the dependency role is not connected to the current role the moment the dependency appears, the resolution of the

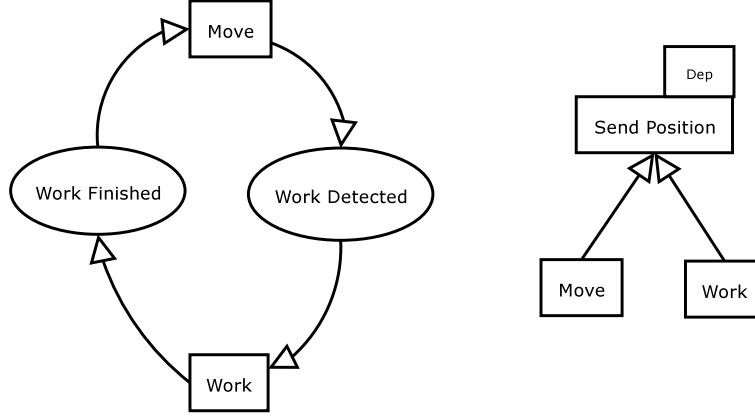


Figure 1: Example of a simple behavior graph on the left, boxes are roles, ellipses are links. On the right the representation of a dependency role.

dependency is delayed until a role connected to the applicable dependency role becomes active.

2.5 Refreshment of information

A final aspect of our basic model concerns the refreshment of information. When new information becomes available to the agent it may switch roles. In the proposed model it is only necessary to update the information in the links that start from the current role. After all, only the goal roles of these links are candidates to become the next current role. This approach not only saves computation time, more important is its impact on the scalability of the model. Since the refreshment of information happens only locally it is independent of the number of roles and the size of the behavior graph.

3 Model for adaptive behavior

In this section we extend the basic role based model towards adaptive agents. An adaptive agent dynamically changes its behavior according to altering circumstances it is faced with. First we introduce dynamic links for behavior graphs, based on factors. Then we zoom in on adaptive factors.

3.1 Dynamic links based on factors

In the basic model a link can be open or closed, allowing role switching or not. Now we extend a link to have a certain *relevance* to switch roles. The relevance is determined through a set of relevant *factors*. A factor models a property of the agent or its perceived environment. An example of a factor is the distance to a target (when the agent is close to the target the factor is high and the other way around). When the link is evaluated each factor returns a value that indicates the current strength of the property of that factor. Based on a mathematical function the agent

then calculates (using the values of all the factors of the link) the relevance to switch roles via that link.

A simple approach is to use a linear sum, i.e. the relevance to switch roles is then:

$$P = \sum_i w_i \cdot f_i$$

where $0 \leq w_i \leq 1$ is the weight of factor i and f_i its current value, with $\sum_i f_i = 100$. A disadvantage of this simple model is that none of the factors is able to dominate the result and so force a role switch along the link. To resolve this problem we can give a link a standard relevance to switch roles. This relevance can then be influenced by the factors as follows: $P = P_{standard} + \sum_i w_i \cdot f_i$. $P_{standard}$ is the pre-defined standard relevance, w_i the weight of factor i and $-100 \leq f_i \leq 100$ the value of the factor. This method is more flexible, e.g. it supports the negative influence of a factor. However it suffers from another problem: saturation. A link with a relevance of 99 % has almost the same relevance to switch roles as a link with a relevance of 400 %. More complex calculations can also be used, such as Boltzmann exploration (L.M. Kaelbling and Moore, 1996).

Based on the relevances of all links that start from the current role (and for which the preconditions of the goal role hold) the agent then calculates to which role it will switch. The most simple approach selects the link with the highest relevance, alternatively a stochastic probability over the candidate links can be used to calculate a decision.

3.2 Adaptive factors

As stated above, factors model conditions to switch roles via one of the candidate links. We introduce two kind of factors, pre-defined and self-learning factors.

Pre-defined factors allow a designer to express relative preferences for role selection. The definition of pre-defined factors requires a thorough knowledge of the

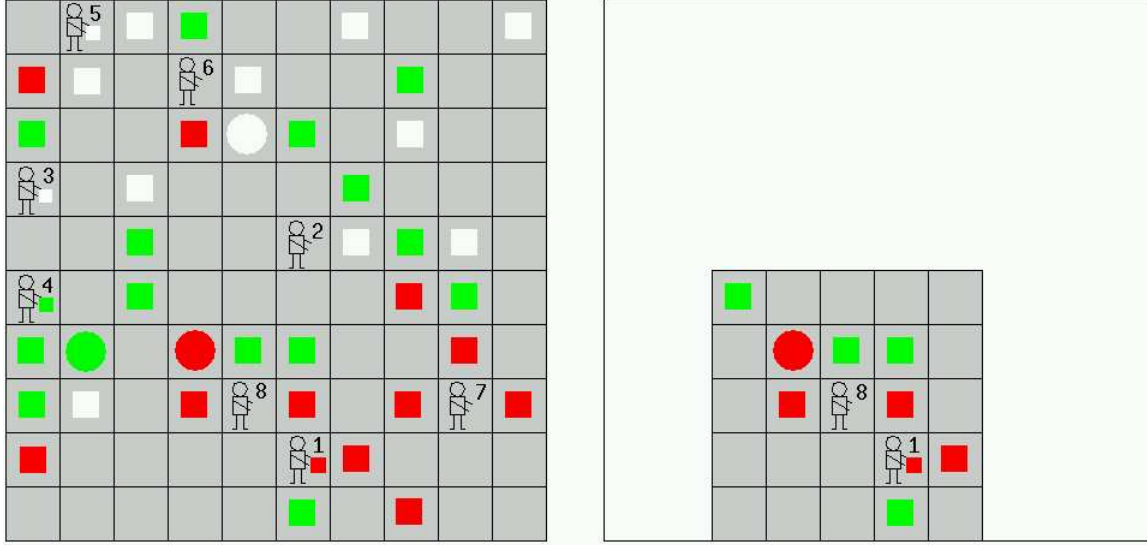


Figure 2: Example of the Packet-World

problem domain. The designer has to identify the relevant properties of the agent or its surrounding environment that affect its role selection. Based on the values of the factors in the links that start from the current role, the agent dynamically calculates to which role it will switch.

Self-learning factors go one step further and allow an agent to adapt its behavior over time. Self-learning factors take into account the good or bad experiences of recent role selections. The result of a role selection is determined by the past performance of the goal role. The calculation of the performance can be done locally, by the goal role itself (e.g. the success of getting an answer to a request) or globally, i.e. a general evaluation function can be used that takes into account macroscopic information (e.g. an agent that follows an alternative route to a target to avoid a blockade). During refreshment, the result of the performance calculation is returned back to the link that uses it to adjust the values of its self-learning factors.

Sometimes however the result of actions is not immediately available, e.g. the answer to a question sent to another agent may be delayed. Meanwhile the agent may have left its previous role. To deliver late results at the right links we introduce a *graph manager*. The graph manager is a module internally to the agent that handles the delivery of late results. Self-learning factors can subscribe themselves at the graph manager when they expect a late result of a goal role. As soon as the result is available the goal role returns it to the graph manager. During the next refreshment, the graph manager passes the result to the correct link which updates the self-learning factors to which the result applies.

Contrary to most traditional models for adaptive selection of behavior that dynamically selects a behavior based on the calculation of a probability value in function of the observed state for each *individual* behavior, self-learning factors enable an agent to construct logical *relations* be-

tween different roles in order to dynamically form paths of roles that are suitable for the current state. When the circumstances change, the agent dynamically adjusts its self-learning factors and shifts its paths of roles accordingly.

For example, in Q-learning, a value is learned for every possible action in a certain state. In our model, every role represents an abstract action or group of actions, that are linked with each other using various kinds of preconditions on the state. In a sense, whereas Q-learning links all states through an action - value pair, we link actions through a condition on the state. Thanks to the introduction of factors, we are able to change the influence of a certain property of the environment on the agent's actions adaptively. Because actions are linked, it is easier to learn paths of actions, rather than paths of states.

In one extreme, the behavior graph can be completely linked, so that the agent can learn all useful paths himself during execution. However, usually some human designer knowledge is already encoded in the graph, so that obviously useless links can be avoided beforehand.

4 The model applied to the Packet-World

In this section we apply the model for adaptive behavior to the Packet-World. First we introduce the Packet-World application and describe a basic behavior graph for agents in the Packet-World. Next we discuss the problem of the 'sparse world' in the Packet-World. To resolve this problem, we extend the basic behavior graph to enable the agents to adapt their behavior when the world gets sparse. Finally, we show simulation results that demonstrate the improved performance of the adaptive agents over the basic agents for a sparse world.

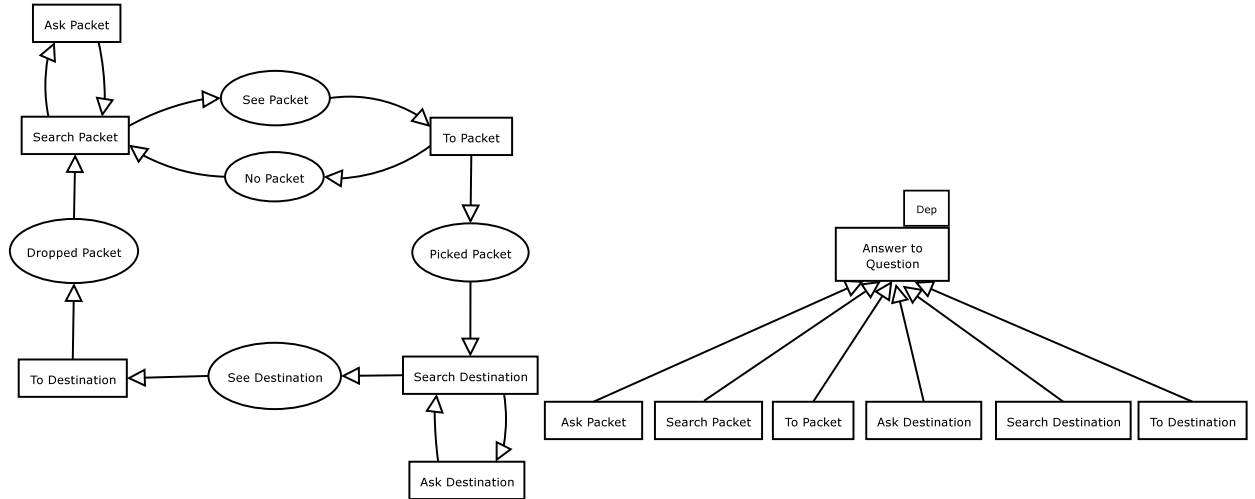


Figure 3: A behavior graph for a basic agent in the Packet-World. On the right the representation of the dependency role to answer to questions.

4.1 The Packet-World

The Packet-World, Huhns and Stephens (1999) Weyns and Holvoet (2002), consists of a number of different colored packets that are scattered over a rectangular grid. Agents that live in this virtual world have to collect these packets and bring them to their correspondingly colored destination. We call a *job* the work of the agents to deliver all packets in the world. The left part of Fig. 2 shows an example of a Packet-World with size 10 in which 8 agents live. Colored rectangles symbolize packets that can be manipulated by the agents and circles symbolize destinations.

In the Packet-World agents can interact with the environment in a number of ways. An agent can make a step to one of the free neighbor fields around him. If an agent is not carrying any packet, it can pick up a packet from one of its neighbor fields. An agent can put down a packet it carries at one of the free neighbor fields, or of course at the destination point of that particular packet. Finally, if there is no sensible action for an agent to perform, it may wait for a while and do nothing. Besides acting into the environment, agents can also send messages to each other. In particular agents can request each other for information about packets or the destination for a certain color of packets.

It is important to notice that each agent of the Packet-World has only a limited view on the world. The view-size of the world expresses how far, i.e. how many squares, an agent can 'see' around him. The right part of Fig. 4.1 illustrates the limited view of agent 8, in this example the view-size is 2.

We monitor the Packet-World via two counters that measure the efficiency of the agents in performing their job. A first counter measures the energy invested by the agents. When an agent makes a step without carrying a packet it consumes one unit of energy, stepping with a

packet requires two units of energy. The energy required to pick up a packet or to put it down is also one unit. Finally, waiting and doing nothing is free of charge. The second counter measures the number of sent messages. This counter simply increments for each message that is transferred between two agents. The overall performance can be calculated as a weighted sum of both counters.

4.2 A behavior graph for agents in the Packet-World

Fig. 3 depicts a behavior graph for a basic agent in the Packet-World. The behavior of a basic agent is simple, it moves to the nearest packet, picks it up and brings it to its destination. If the agent does not perceive any packet (or the destination it is looking for), it searches randomly. However, if it perceives another agent it asks if the other agent knows the target.

We translated this simple behavior into six roles. In the role *Search Packet* the agent randomly moves around looking for packets. However, if it perceives other agents it switches to the role *Ask Packet* requesting the farthest agent whether it perceives a packet. Perceiving another agent is modelled as a precondition of the *Ask Packet* role. The link to switch from the *Search Packet* role to the *Ask Packet* role has no condition. We call such a link a *default link*. A default link is indicated by an arrow without an oval with a condition. The accompanying number ($0 \dots 1$) denotes the chance for an agent to switch roles via that link¹. A link with a chance 1 will always be open. A link with a chance of 0.33 indicates that if all other conditions hold, the agent changes roles via that link in only 33 % of the cases. When the requested agent receives the question, it switches, independent of its current role, to the *Answer to Question* role to give the re-

¹The default value for a chance is 1, this value is not annotated.

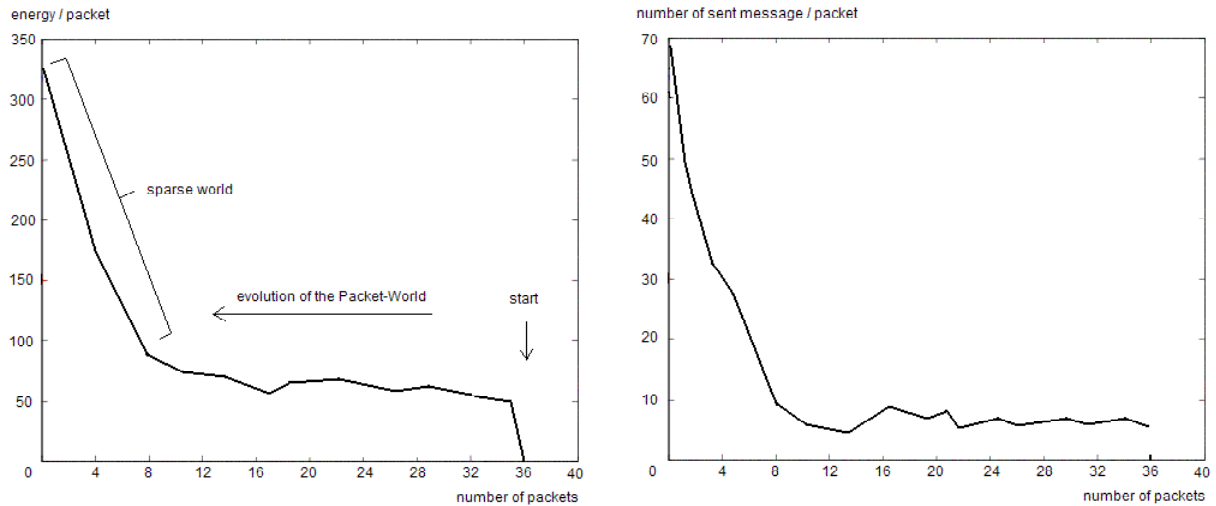


Figure 4: On the left, energy usage during the evolution of the Packet-World, on the right, the number of sent messages.

requesting agent an answer. Since the requesting agent *depends* on the requested agent, the *Answer to Question* role is modelled as a dependency role. When the requested agent receives the location of a packet, it switches via the *See Packet* link to the role *To Packet*, otherwise it continues to execute the *Search Packet* role. In the role *To Packet* the agent moves straight on to the packet. If meanwhile the packet is picked up by another agent, the agent returns via the *No Packet* link to the *Search Packet* role to look for another packet. If it perceives other packets it switches again to the *To Packet* role and moves to the nearest perceived one. If no other packet is visible the agent keeps searching for a packet in the *Search Packet* role. Finally when the agent succeeded in picking up a packet it switches via the *Picked Packet* link to the *Search Destination* role. In a similar way the agent then searches for the destination of the packet it carries. When the agent finally delivers the packet it enters the *Search Packet* role to look for the next packet.

4.3 The problem of the sparse world

We observed the behavior of the basic agents in the Packet-World and noticed a particular problem for the agents. When the agents deliver packets at the destination the number of remaining packets in the environment decreases. When the world gets sparse, i.e. when only a couple of packets are left, we observed that the behavior of the agents becomes inefficient. We called this the 'sparse world' problem.

Fig. 4 shows simulation results that illustrate the effects on energy consumption and communication traffic when the world gets sparse. As the number of packets in the environment decrease, the graphs have to be read from the right to the left side. The left graph shows the energy used by the agents for each packet that is delivered.

From the start (in this case the number of initial packets was 36) until approximately 8 packets the energy consumption is fairly constant. However the required energy to deliver the remaining packets strongly increases. The right graphs shows the number of messages sent for each packet that is delivered. Similar to the energy consumption, the number of sent messages increases remarkable when the number of remaining packets becomes less than 8.

From the observations, we identified three kinds of problems for the agents when the world got sparse:

1. The number of requests for packets explodes while most agents can not give a meaningful answer. We call this the *request explosion* problem.
2. Most agents keep searching aimlessly for packets wasting their energy. We call this the *energy waste* problem.
3. When several agents detect one of the few packets all of them run at it while in the end only one agent is able to pick it up. We call this the *storming* problem.

4.4 A behavior graph for adaptive agents

To resolve the problems with the sparse world, we designed an adaptive agent according to the model described in section 3. Fig. 5 depicts the behavior graph of the adaptive agent.

We reused the roles of the basic agent for the adaptive agent. We added two extra waiting roles, *Wait to Search* and *Wait to Collect*. Similarly as to the basic agent, the adaptive agent has a dependency role *Answer to Question* to answer questions sent by other agents (this dependency role is not depicted). As for the basic agent, the adaptive agent is able to answer questions

to the basic behavior we introduced the *Wait to Collect* role. The *Spot Same Target* link from the *To Packet* role to the *Wait to Collect* role is influenced by the behavior of the other agents within the perceptual scope of the agents. The *Spot Same Target* link contains two dynamic factors: *Same Target* and *Nearest to Target*. The *Same Target* factor increases the chance to switch to the *Wait to Collect* role when the agent suspects that other agents move to the packet it is looking for. Therefore the agent compares (in the *To Packet* role) for each visible agent the distance to the packet it is looking for with the distance to the nearest (visible) packet for the other agent. The second factor *Nearest at Target* decreases the chance to switch to the *Wait to Collect* role when the agent believes it is nearer to the targeted packet than the other agents inside its perceptual scope. Therefore the agent simply compares its own distance to the target packet with the distance of each visible agent to the packet. From the *Wait to Collect* role the agent returns to the *Search Packet* role via a default link. In Fig. 5 the chance to switch back is set to 0.3. The *Wait to Collect* role enables the agent to deal with the storming problem. Especially when only a few packets are left, the factors in the *Spot Same Target* link work very efficient.

When the agent picks up a packet it switches to the *Search Destination* role. From then on, until the packet is delivered, the adaptive agent behaves the same way as the basic agent does.

4.5 Simulation results

To verify whether the adaptive agents behave as desired we did two types of simulations. First we compared the behavior of the basic agents with the adaptive agents in a sparse world. Then we put the agents in a homogeneous world and looked at the behavior of both types of agents when the world gets sparse.

4.5.1 Simulation results for a sparse world

In this simulation we are only interested in the behavior of the agents with respect to the problems of the sparse world. Fig. 6 depicts the sparse world of the first simulation. The environment size is 45x45. In the MAS there are 3 colors of packets, with for each color 3 packets. Packets and destinations are positioned such that agents with a large perceptual scope perceive a couple of packets and the corresponding destination. Packets are located far enough from the agents to clearly distinguish the energy consumption for both types of agents. Furthermore, we clustered the agents to accentuate possible storming behavior.

Fig. 7 compares the energy usage and communication traffic for both types of agents. The depicted graphs represent the average results of energy consumption (on the left) and requests for packets (on the right) for 80 runs. The right figure illustrates the significant reduction of re-

quests for packets for adaptive agents. The left figure illustrates the decreased energy consumption to deliver the packets. The simulation results demonstrate the improved behavior of the adaptive agents in the sparse world. The adaptation of the behavior appears very quickly, in the example the effects of adaptation are already noticeable after two of the last 8 packets where delivered.

4.5.2 Simulation results for a homogeneous world

In this section we show how adaptive agents change their behavior while the environment changes and the world becomes sparse. Fig. 8 depicts the test world we used in this simulation.

In the environment of size 32x32 we put 25 packets of 2 different colors, homogenously scattered. 10 agents spread over the world have to collect the packets and bring them to the correct destination.

The average simulation results for 80 runs are depicted in Fig. 9 (in the graphs only the results for the collection of the last 40 packets are depicted). The figure shows that adaptation starts to work when approximately 10 packets are left in the environment. From that point on, the energy consumption as well as the communication is significantly lower for adaptive agents. The simulation results demonstrate that adaptive agents recognize the changes in the environment and change their behavior accordingly. As an overall result, we calculated an expected gain of 12 % for the adaptive agents in the second simulation.

5 Conclusions

In this paper we proposed a role based model for adaptive agents. The model describes the behavior of an agent as a graph of roles. Adaptivity is achieved through factors in the links of the graph. Pre-defined factors express relative preferences for role selection. Self-learning factors reflect the extent of success of recent role selections, enabling an agent to dynamically form paths of roles that are suitable for the current state. Adapting the logical *relations* between different roles contrasts to most existing approaches for adaptive behavior selection that dynamically selects a behavior based on the calculation of a probability value for each *individual* behavior. The model is similar to a reinforcement learning approach (L.M. Kaelbling and Moore, 1996), with the notable difference that it allows a designer to logically group large numbers of state together in a role, and thus avoids a state-space explosion. It also allows easy integration of "common knowledge" through the pre-defined factors.

A prerequisite to apply the proposed model is sufficient knowledge of the domain. To design a behavior graph, the designer must first be able to identify the roles for the agents and all the possible and necessary switches between roles. Binding conditions for a role to become active have to be modelled as pre-conditions, dependencies between roles are modelled as separate dependency

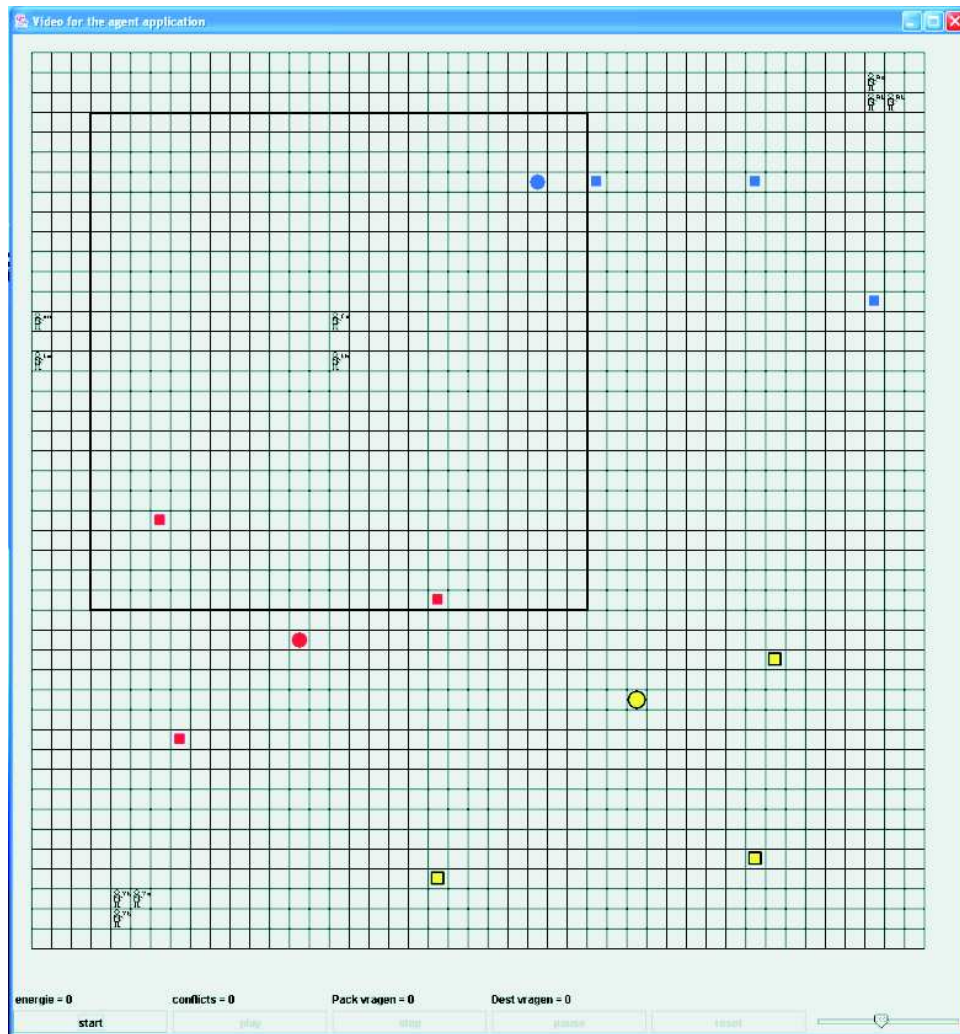


Figure 6: The sparse world used in the first simulation.

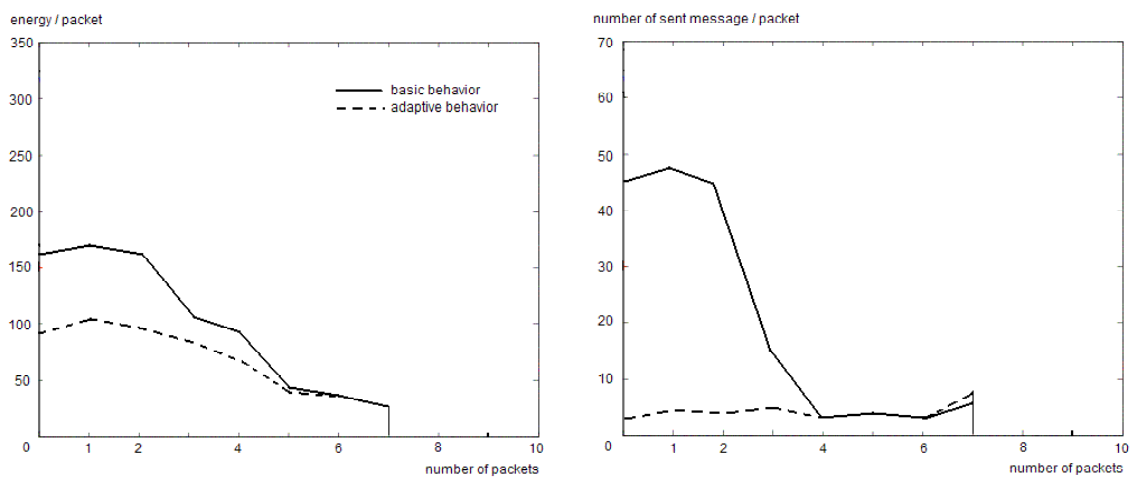


Figure 7: Comparison non-adaptive versus adaptive agent for a sparse world. The left figure denotes the energy consumption vs. the number of packets, the right figure the number of sent messages vs. the number of packets.

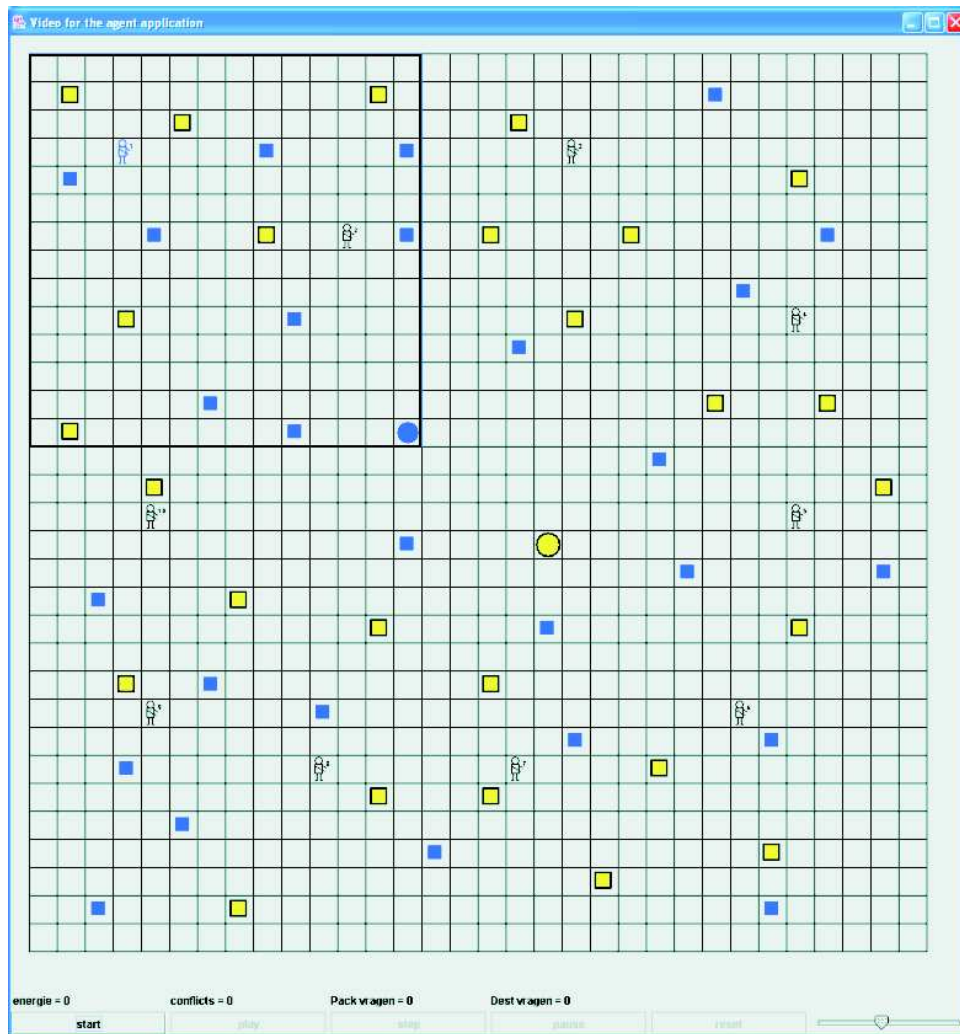


Figure 8: The homogeneous world used in the second simulation.

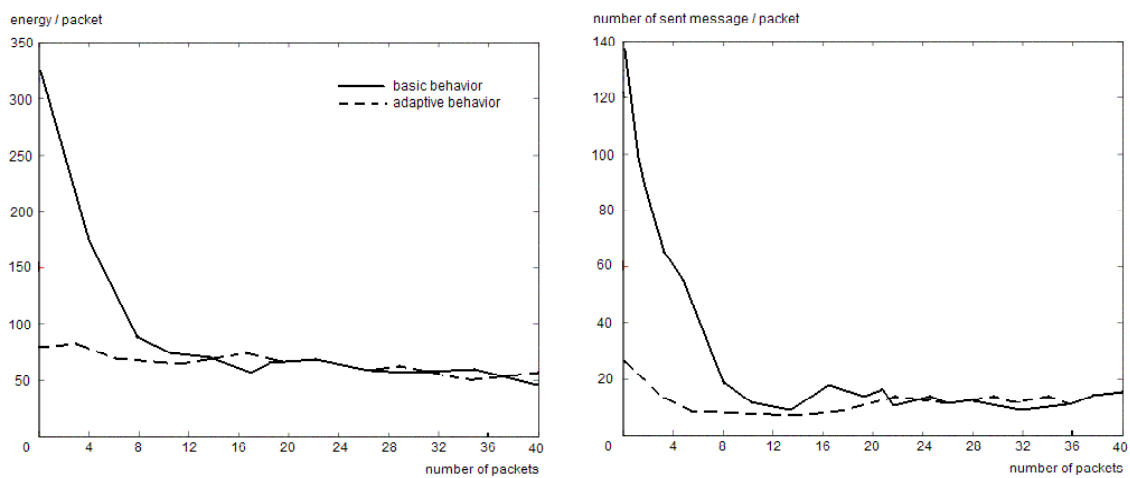


Figure 9: Comparison non-adaptive versus adaptive agent for a homogeneous world. The left figure denotes the energy consumption vs. the number of packets, the right figure the number of sent messages vs. the number of packets.

roles. Second, to identify the factors in the links, the designer has to identify the relevant properties for the agent to switch roles. And finally, to define self-learning factors, the designer needs a notion of the relevance of the performance of roles. On the other hand the designer does not have to manage the problems all alone. Adaptivity adds value by adjusting the behavior of the agent at runtime according to the changing (not explicitly foreseen) circumstances.

To illustrate that the model works, we applied it to the Packet-World. We discussed simulation results that demonstrate that the model enables the agents to adapt effectively to a changing environment.

References

- E. Bonabeau, F. Henaux, S. Guérin, D. Snyers, P. Kuntz, and G. Theraulaz. Routing in telecommunications networks with “smart” ant-like agents. In *Intelligent Agents for Telecommunications Applications '98 (IATA'98)*, 1998.
- A. Drogoul. De la simulation multi-agent a la résolution collective de problèmes. Ph.D thesis, l' Université Paris 6, France, 1993.
- A. Drogoul and J.D. Zucker. Methodological issues for designing multi-agent systems with machine learning techniques: Capitalizing experiences from the robocup challenge. Technical Report 041, LIP6, 1998.
- M.N Huhns and L.M. Stephens. Multi-agent systems and societies of agents. Ed. G. Weiss, *Multi-agent Systems*, MIT Press, 2, 1999.
- L.M. Littmann L.M. Kaelbling and A.W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4, 1996.
- P. Maes and R.A. Brooks. Learning to coordinate behaviors. *Autonomous Mobile Robots: Control, Planning and Architecture*, 2, 1991. IEEE Computer Society Press, CA.
- K. Schelfhout and T. Holvoet. To do or not to do: The individual's model for emergent task allocation. In *Proceedings of the AISB'02 Symposium on Adaptive Agents and Multi-Agent Systems*, 2002.
- D. Weyns and T. Holvoet. The packet-world as a case to investigate sociality in multi-agent systems. Demo presented at the 1th International Conference on Autonomous Agents and Multiagent Systems, AAMAS'02, Bologna, 2002.
- T. De Wolf and T. Holvoet. Adaptive behavior based on evolving thresholds with feedback. In D. Kudenko D. Kazakov and E. Alonso, editors, *Proceedings of the AISB'03 Symposium on Adaptive Agents and Multiagent Systems*, 2003.

Dynamic scheduling of multiple agents for a heterogeneous task stream

Neil Windelinckx and Malcolm Strens*

{nwindelinckx,mjstrens}@QinetiQ.com

*Future Systems Technology Division, QinetiQ
G020/A9, Cody Technology Park, Farnborough, Hants. GU14 0LX. U.K.

Abstract

We consider a class of problem in which a group of (physical) agents must continuously perform tasks that arise at arbitrary locations across a large space. To complete each task, one or more agents will be required to move to its location, and stay there for a period of time. A hybrid scheduling algorithm is derived in which proposed plans are evaluated using a combination of short-term lookahead and a value function acquired by trial-and-error learning with a simulation (reinforcement learning). We demonstrate that the dynamic scheduler can learn not only to allocate agents to tasks efficiently, but also to position the agents appropriately in readiness for new tasks, and conserve resources over the long run.

©QinetiQ Ltd, 2004.

1 Introduction

Dynamic scheduling is a resource allocation problem in which the plan must be reviewed constantly in response to exogenous events (*e.g.* user-specified tasks entering the system). A plan consists of an ordered allocation of agents to tasks, together with any associated parameters.

A “heterogeneous task stream” contains tasks of several types (possibly from many different users/clients). The tasks may have individual constraints on their successful completion (*e.g.* earliest or latest start or finish times), or joint constraints (*e.g.* completion of task A before starting task B).

This paper describes a part of a proprietary QinetiQ software suite for dynamic scheduling in multi-agent systems. We focus here on the case where the resources are physical agents (vehicles or robots) and the tasks arise at various locations across the space in which they operate. For example, an on-demand transport system consisting of many individual road vehicles will be required to review its planned schedule whenever a new journey request enters the system.

We view the problem as one of optimal control (sequential decision making) in a partially observable environment. Boutilier et al. (1999) survey of existing work in this topic area, which links operations research (OR), reinforcement learning (RL) and planning. This contrasts the usual formulation of multi-robot task allocation (MRTA) as a static optimization problem Gerkey and Mataric (2003).

The partial observability arises from not being able to predict what new tasks will arise in the stream of tasks

that is provided to the agents. Therefore it is not possible to *plan* beyond the tasks that are already known. To obtain a near-optimal solution requires the agents not only to complete the known tasks as rapidly as possible, but also to remain well-positioned for new tasks, and to preserve resources (*e.g.* fuel or battery power).

From a decision-theoretic viewpoint, the state of the system is high-dimensional, because it includes the state of each agent, the known tasks, and the distribution of unseen tasks. The actions available to the decision-maker are assignment of agents to tasks. The cost function that is to be optimised consists of the accumulated “rewards” received for successful completion of tasks.

1.1 Paper outline

Section 2 introduces sequential decision theory formulations of the problem, and identifies the need for approximate solution methods. Section 3 describes the hybrid planning/learning approach and its underlying assumptions. Section 4 provides an evaluation of the hybrid approach, comparing with short-term planning and a heuristic approach. This is followed by a discussion (section 5) and conclusions (section 6).

2 Theoretical foundations

A formal description is given of the Markov Decision Process (MDP) model that underlies dynamic programming, reinforcement learning and optimal control. Models for partial observability are introduced, and the need for approximate solution methods is identified.

2.1 Markov Decision Process

An MDP is a discrete-time model for the stochastic evolution of a system's state, under control of an external input (the agent's action or agents' joint action). It also models a stochastic reward that depends on the state and action.

Definition A Markov Decision Process is given by $\langle X, A, T, R \rangle$ where X is a set of states and A a set of actions. T is a stochastic transition function defining the likelihood the next state will be $x' \in X$ given current state $x \in X$ and action $a \in A$: $P_T(x'|x, a)$. R is a stochastic reward function defining the likelihood the immediate reward will be $r \in \mathbb{R}$ given current state $x \in X$ and action $a \in A$: $P_R(r|x, a)$.

2.2 Optimal policy

Combining an MDP with a deterministic control policy $\pi(x)$, that generates an action for every state gives a closed system. A useful measure of policy performance, starting at a particular state, is the expected value of the *discounted return*:

$$r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

$\gamma < 1$ is a discount factor that ensures (slightly) more weight is given to rewards received sooner. For a given policy π and starting state x , we write $V_\pi(x)$ for this expectation. If a trial has finite duration, then the infinite sum is truncated, and we can allow the case $\gamma = 1$ (finite horizon undiscounted return).

A very useful quantity for deriving optimal control policies is the state-action value $Q_\pi(x, a)$ which is the expected discounted return in state x when taking action a initially, then following policy π for action selection thereafter. Given an MDP and a discount factor, there exist optimal value functions Q^* and V^* that satisfy:

$$V^*(x) = \max_a Q^*(x, a)$$

$$Q^*(x, a) = R(x, a) + \gamma \sum_{x'} P_T(x'|x, a) V^*(x')$$

where $R(x, a)$ is the expected instantaneous reward for action a in state x , determined by $P_R(r|x, a)$. The optimal policy is to select the action a that maximises the state-action value:

$$\pi^*(x) = \arg \max_a Q(x, a)$$

2.3 Reinforcement learning

Dynamic programming (Bellman, 1957) is a simple process for finding π^* for a *known* MDP, using repeated updates such as:

$$Q'(x, a) \leftarrow R(x, a) + \gamma \sum_{x'} P_T(x'|x, a) \max_{a'} Q(x', a')$$

This causes Q to converge to Q^* when applied repeatedly in every state-action pair (x, a) .

Reinforcement learning (RL) attempts to find the optimal policy for an *unknown* MDP by trial-and-error interaction with the system (Sutton and Barto, 1998). *Model-based* RL methods estimate the MDP, then solve it by dynamic programming. “*Conventional*” reinforcement learning methods such as Q learning (Watkins, 1989) and SARSA estimate Q^* (and hence π^*) directly without estimating P_T explicitly. *Policy search* reinforcement learning methods parameterise π and search directly for values of these parameters that maximise trial returns (e.g. Strens and Moore (2002)).

2.4 The need for approximate solution methods

Partially observable problems in which the agents cannot observe the full system state are much more challenging. These can be formulated using an extended model called a *partially observable* Markov Decision Process (POMDP) that adds an observation process to the MDP. The optimal policy can be derived in the same way as for the fully observed case (above), but replacing the MDP's state-space with the POMDP “belief space” (Martin, 1967). An agent's *belief* is its estimate for the probability density over the true system state, given the observation history. Unfortunately the set of beliefs has infinite size, and so methods that enumerate the state space (e.g. dynamic programming and Q learning) cannot be applied directly. Policy search, however, remains feasible.

In dynamic scheduling domains the partial observability usually arises not from noisy measurements of the physical system state, but in the unknown tasks that have not yet been presented to the agents. Even without this uncertainty, approximate solution methods would be required, because the state space is continuous and high dimensional. (The state consists of agents' physical states and the known task descriptions.) The MDP and POMDP models that are so useful in theory are too general, in practice, to find solutions to even the simplest dynamic scheduling problem. However, they offer a “perfect solution” against which approximate methods can be judged.

3 Hybrid planning and learning algorithm

Although partial observability and large state spaces make dynamic scheduling a very difficult problem, it should be possible to simplify the solution by exploiting specific problem structure. It is natural to attempt to *factor* the “full” MDP representing the multi-agent, multi-task system into many smaller sub-problems that can be individually solved. It is also appropriate to exploit as much prior knowledge about the problem as possible, to ensure best use is made of (trial-and-error learning) simulation runs.

This prior knowledge will be presented to the system in the form of “task models” that predict the time taken to perform known tasks, and “state features” that compress the state-space into a compact description for learning.

3.1 Short-term predictability assumption

The special form of partial observability found in dynamic scheduling leads to an approach in which we assume that the short-term future is *predictable* and can be treated as a static planning problem, but the long-term future is unpredictable and requires an approximate method that accounts for uncertainty. Hence we propose a hybrid planning/learning system in which short-term planning ignores the exogenous events (new tasks entering the system) up to some “planning horizon”, but a value function is learnt to represent the medium and long term benefits of being in a particular joint state when the planning horizon is reached.

This assumption allows replanning to be viewed as a search over a discrete set of plans (ordered allocations of agents to tasks). A planning mechanism must be made available that takes as input the current joint state x (of the agents) and a proposed plan P , and yields an end state x' and an expected discounted return $R(x, P)$, up to the planning horizon τ . The total value of the plan is given by:

$$Q(x, P) = R(x, P) + \gamma^\tau V(x')$$

where $V(x')$ is a state-value function that will account for the “goodness” of being in state x' ; *i.e.* a prediction for the expected discounted return from the planning horizon to the end of the trial. As we expect predicted rewards to become less reliable with time, another discount factor, λ can be introduced to discount more severely (in short-term planning). Suppose plan P predicts that reward r will be obtained at time t (from the time at which the plan is formed). The contribution to $R(x, P)$ is now given by $(\gamma\lambda)^t r$. In the experiments here, we used $\gamma = 1$ (undiscounted finite horizon return) and $\lambda = 0.9986$.

Normally it will not be possible to evaluate every possible plan, but many methods are available for searching over this discrete space. (Our search method is based on making local changes to the current plan.)

3.2 The weak-coupling assumption

Once an allocation decision (of agents to tasks) has been made, the tasks essentially become independent in terms of rewards received and end states (Meuleau et al., 1998). For example, if agent A is assigned to task 3 and agent B to task 5, the effectiveness of A in task 3 is assumed to be independent of the individual steps taken by B in performing task 5. (This assumption cannot be made in all scheduling domains.) This suggests a two-level planning process in which the top level (allocation of agents to tasks) is “globally” aware, but the lower level planning

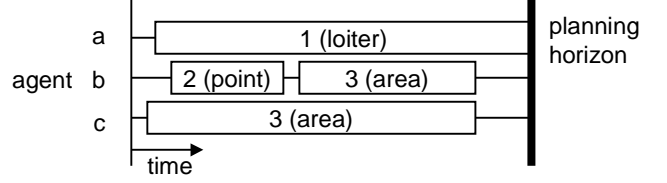


Figure 1: A plan is an assignment of tasks to agents.

(execution of individual tasks) can be achieved in a *local* context (a particular agent and particular task).

Task performance models provide the lower-level planning capability. Given a particular allocation decision (agent to task), the task performance model predicts both the time taken to complete the task, and the likely final location of the agent after completion. The task performance model is also responsible for predicting the rewards (performance feedback signals) that will be obtained on task completion. Using these performance models, it is possible to estimate the short-term rewards that will accrue from a particular assignment of tasks to agents. Such an assignment is called a “short-term plan” (Figure 1).

Strictly, a task performance model takes as input the state of a single agent x_j , the parameters y_k describing a particular task instance (k), and any allocation parameters α_{jk} . It provides as output a new state x'_j , a duration t_{jk} and an expected discounted return r_{jk} . More generally the task model can sample from stochastic outcomes:

$$P(x'_j, t_{jk}, r_{jk} | x_j, y_k, \alpha_{jk})$$

For each agent, a series of these predictions can be performed to plan for the ordered list of tasks that are assigned to it. If a prediction takes the total time beyond the planning horizon, then the actual end state must be interpolated. By appropriately discounting and summing the individual returns ($\{r_{jk}\}$) associated with each task allocation in a plan P , the short-term value $R(x, P)$ of that plan is obtained. For tasks that require agents to cooperate, *group* task models can be used to predict the outcome of assigning multiple agents to a single task.

3.3 Greedy short-term plans

A “greedy” planning approach makes use of the approximate task performance models to predict the short-term return $R(x, P)$ that will accrue from a particular plan P . Figure 2 illustrates how instantaneous rewards (solid blocks) may be predicted by the task models. The prediction is performed only up to the planning horizon. These are summed¹ over all agents to give a reward profile (lower chart in the figure). The area under this profile measures the expected return for the short-term plan.

¹The rewards should first be discounted if $\gamma\lambda \neq 1$.

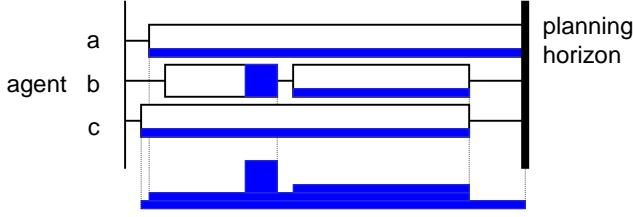


Figure 2: Greedy planning.

The greedy planning approach is to search for an assignment of tasks to agents that maximises the short-term expected return. This is better than simply selecting the closest agent for each new task. However, greedy planning is not optimal because it ignores several important factors in decision-making: a. the need for positioning of the agents in readiness for new tasks; b. the preservation of energy and other resources (*e.g.* robot battery life); c. the approximate nature of the task performance models.

In the greedy planning, we can also make use of a set of default behaviours that can be selected when no tasks are scheduled. These allow the agents to reposition ready for new tasks to enter the system. There are no rewards associated with performing the default behaviours (but they will often be selected because they have an influence on the position value function at the planning horizon).

3.4 Value function learning

Positioning of the agents (“readiness”) is essential in problem domains where there are tight time constraints on the completion of tasks. This means that agents should be positioned so as to minimise the time taken to reach new task locations. Resources have a value that depends on the type of problem domain. In long-endurance domains, learning strategies that minimise energy consumption may be essential for effectiveness. If agents are operating in a group, there will often be an additional benefit to balancing the use of resources as equally as possible between them during a trial.

The representation for learning is a state-value function, that will be evaluated at the planning horizon indicated in Figure 1. The value function provides an approximate mapping from the state (of the agents) to the expected discounted return from that time onward. For example states in which the agents are well-positioned (and have full energy stores) will have the highest values. The value function is always evaluated at the planning horizon (a time in the future), rather than the current time. This simplifies the decision-making process because the known tasks are assumed to have no influence beyond the planning horizon. Therefore it is the predicted state x' of the agents, rather than the status of *tasks*, that determines the state-value $V(x')$ at the planning horizon.

Learning takes place at the end of each trial, and aims to reduce (by gradient descent) the residual error between

the actual return received during the trial and predictions that were made during the trial using the value function. The gradient descent rule is simple to derive for the linear function approximator used here (*e.g.* Sutton and Barto (1998)), and for nonlinear generalisations such as the multilayer perceptron. Training instances are obtained each time replanning takes place. This occurs whenever a new task enters the system or an existing task is completed, and at regular time intervals during the trial. (Other learning rules can be considered if the function approximator or task models are known to be inaccurate: direct policy search or policy gradient descent are more robust, but potentially slower.)

Suppose replanning has taken place N times during a trial. Let $(i = 1, \dots, N)$ index these instances. The predicted value (for the chosen plan) is given by the weighted sum of m state features:

$$v_i = \sum_{j=1}^m w_j x_j^{(i)}$$

where $x_j^{(i)}$ is state feature j for instance i . Suppose that the recorded discounted return (from that instance onward) is r_i . Then the residual error is given by:

$$e_i = (v_i - r_i)$$

Summing the squared error over instances, and differentiating with respect to the weights yields a steepest descent direction in weight-space. This leads to an update rule:

$$w'_j \leftarrow w_j - \beta \frac{1}{N} \sum_{i=1}^N e_i x_j^{(i)}$$

The learning rate β for the feature weights was decayed during the experiments, to force convergence of the value function.

4 Evaluation

Evaluation of the hybrid planning/learning approach requires a high-speed simulation that allows repeated trials to be performed (for value-function learning). We describe the simulation software that achieves this, a set of state features, a challenging test scenario with 3 task types, and the evaluation results.

4.1 Performance system

An asynchronous discrete event simulation has been developed to support machine learning and optimisation of agent behaviours. In most simulations, the majority of computational cost is involved in the update of the dynamic parts of the system (*e.g.* the agents). Simulations are normally clocked at regular intervals (“synchronously”) to ensure an adequate level of accuracy (and also for simplicity). In contrast, we aim to update the

state of each simulation element only when it is needed for decision-making or graphical output. Where possible the update is performed analytically (*i.e.* by direct calculation) but it may be necessary to apply numerical integration in some cases. An example of analytical update is the calculation of agent paths as a sequence of arcs (corresponding to periods of constant turn rate).

At the core of the model is a scheduler that processes events in the simulation in order. Each event is sent to a simulation element (or higher level process) where some processing is performed. For example, the process controlling an agent in a loiter pattern will send a short sequence of events to the scheduler; each event, when processed by the agent's simulation model, causes it to update its state (position, motion, energy store) to the current time, and the new turn rate (demanded by the event) is set.

Built upon this event handling mechanism is the ability for processes within the simulation to send messages to each other. These messages are themselves events, but they represent the actual information that would be passed across a communications network (or bus) within the operational system. Delays and failure modes can also be simulated. Therefore the simulation is built up from a set of processes that operate by receiving, processing and sending messages to each other. Some of these processes are used to represent simulated physical systems (the agents) and others form part of the operational software.

The system has a graphical user interface which allows basic control of the learning system. Although learning trials are normally executed without graphical display, it is possible to inspect occasional trials, through two or three dimensional display, as shown in Figure 3. Here the agent positions are indicated by small triangles. Their short-term plans are indicated by lines. Markers indicate the fixed set of reference locations (w_0 to w_4) and locations of known tasks (all others). Detailed implementation of the performance system (dynamics, low-level control, task constraints, *etc.*) is beyond the scope of this paper.

4.2 State features

The state-value function was chosen to be a linear combination of state features. The state features are intended to represent, in a compact form, the aspects of the agents' state that are relevant for subsequent decision-making. Some are simply averages over the group of agents, whereas others represent relational or relativistic information. A set of reference locations were used to define the operating area for each trial. All the features are listed here:

1. Time left: proportion of maximum trial duration remaining

2. Bias: a constant additive term
3. Mean energy: the average of the agents' energy levels.
4. Max energy: the maximum of the agents' energy levels.
5. Median energy: the median of the agents' energy levels.
6. Min energy: the minimum of the agents' energy levels.
7. Unreadiness: the average (over the reference locations) of the distance of the closest agent to that location.

The 7 values to be learnt are the weights for each of these features in the value function. A positive weight for a feature will mean that situations in which the feature is large are preferred. For example we expect one of the energy features to have a large positive weight to reflect the remaining endurance of the group. The time left and bias features have no effect on decisions between alternative plans, because they are independent of the agents' states, but they can improve the accuracy of the value function.

4.3 Example scenarios

The closed-loop simulation model was configured for repeated trials with 5 agents. The state of each agent is its 2D position, direction of travel, speed and energy level. Each agent is allowed only two possible speeds (40 and 60). For these speeds, energy is consumed at 1 unit per step and 2 units per step respectively. Each agent is able to turn at a maximum constant rate of 0.1 radians per step.

Replanning takes place whenever a new task enters the system, a known task is completed, and at regular intervals (300 steps). The trial time is limited to 10,000 steps. For "endurance" scenarios energy is limited to 10,000 units, but for "non-endurance" scenarios it is essentially infinite.

The planning horizon was 2000 steps and the planning discount factor $\lambda = 0.9986$. At trial number k , the learning rate was chosen to be $\beta = \beta_0 \exp(-0.003k)$, where the initial value is $\beta_0 \equiv 0.85$.

4.4 Three task types

Three types of task have been identified:

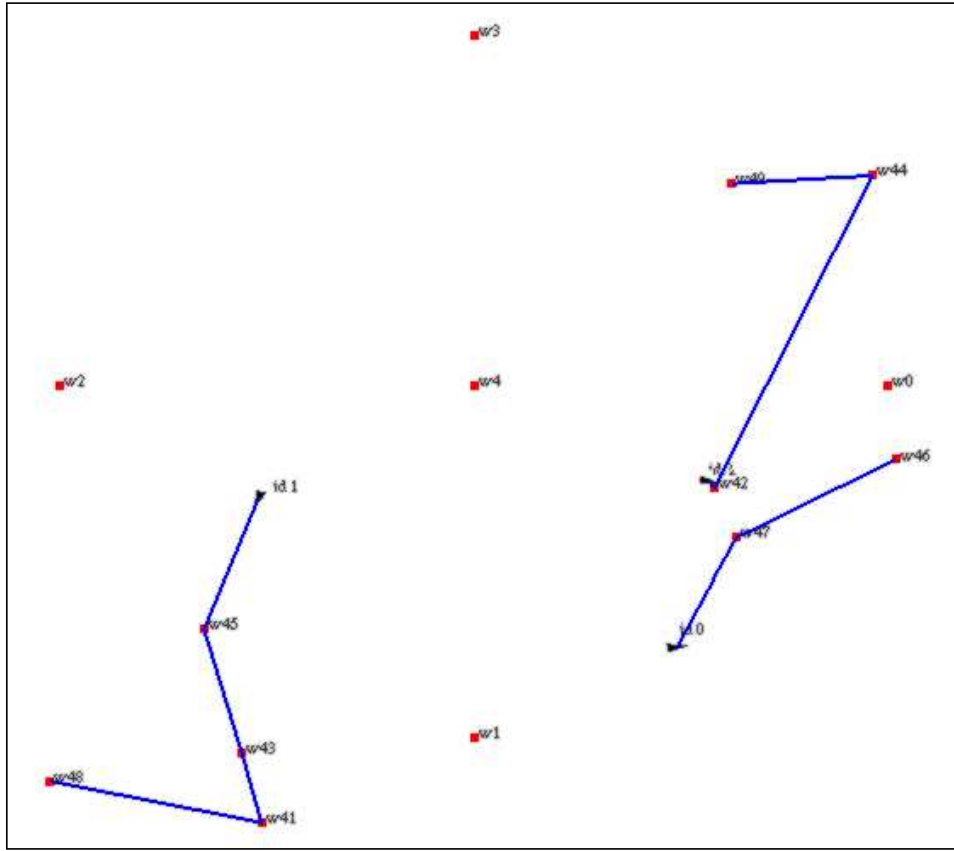


Figure 3: Simulator plan view.

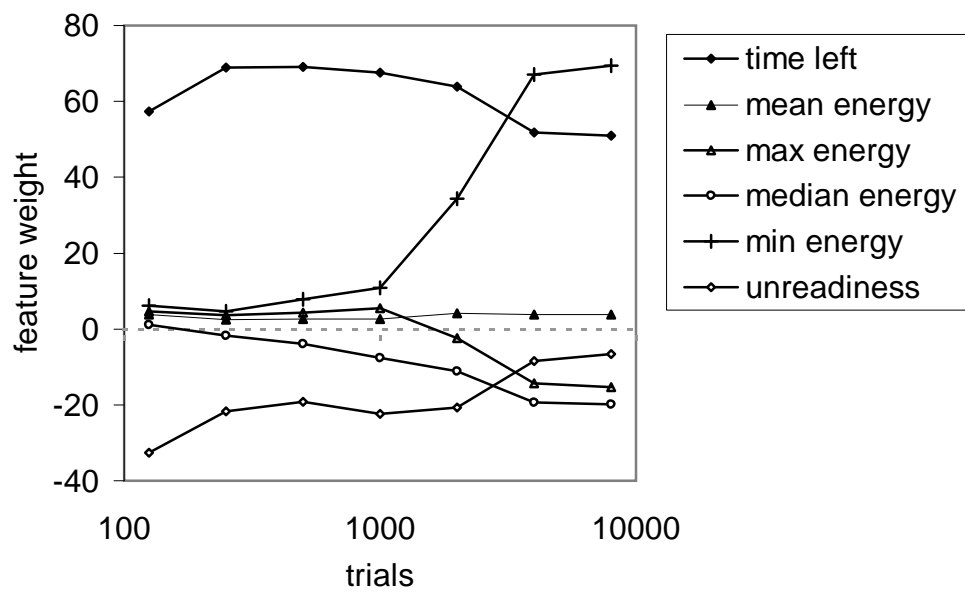


Figure 4: Learnt feature weights: endurance scenario.

Table 1: Major task types, frequencies of occurrence and rewards.

Task type	Frequency	Reward
Point	80%	1
Area	18%	4
Loiter	2%	30

1. Point tasks (*e.g.* sensing): send one of the agents to a location; upon arrival the task is deemed complete.
2. Area tasks (*e.g.* searching): move to a location, then systematically move around the region surrounding that location.
3. Loiter tasks: move to a location and stay there for a period of time.

The specific instance of area task used here, is to visit 4 locations arranged in a unit square; a loiter task required one agent to move to the specified location and stay there (moving in a figure-of-eight pattern) for a specified period of time. The actual task mix (Table 1) is diverse in terms of duration (and rewards) but the three types contribute almost equally to overall trial returns (*i.e.* summed rewards). The default behaviours, also available to the planner, are effectively the same as a slow-speed loiter, in which the agent moves to (then waits at) one of the reference locations.

There is a random delay between successive tasks, given by $1000u^{10}$ where u is uniformly distributed in the range $[0, 1]$. Each task has a 1 in 10 chance of having a precondition. This precondition has equal chance of being (completion of) each of the five preceding tasks. Every task must be completed within a specified time period given by 2^u where u is drawn uniformly in the range $[9, 9.65]$. The location of each task is uniformly distributed across the area of operation (a 100,000-unit square).

4.5 Results

Results were obtained using 8 runs for each approach. Standard errors were computed and are shown as error bars on the plots. First we give results for “endurance” scenarios in which the group of agents may not survive until the end of the trial unless they choose the more efficient (lower) speed and balance effort between themselves. Once one agent has run out of energy, it can play no further part in the trial, and so the remaining agents must travel further (on average) to reach new tasks.

Figure 5 shows (a decaying average of) returns at the end of each trial for three methods: hybrid planning/learning system, the greedy planning system, and

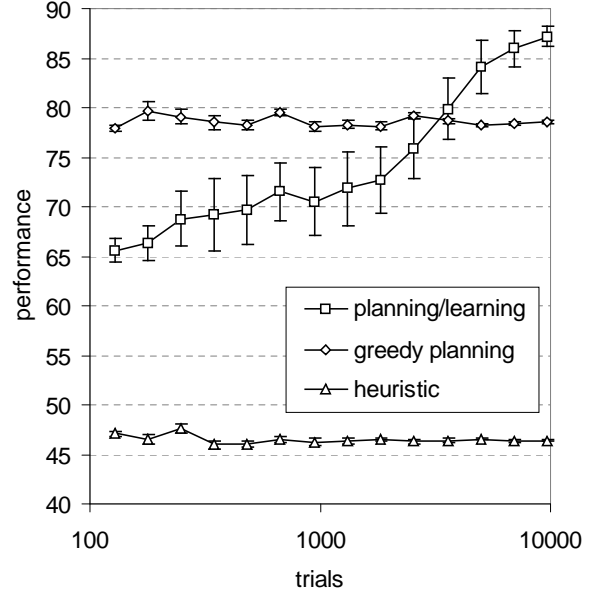


Figure 5: Performance comparison: endurance scenario.

a simple heuristic policy (assign each task to the closest available agent). We observe that short-term planning alone makes a big difference to the efficiency with which the tasks are performed. Note that short-term planning depends strongly on the availability of task performance models: that is the ability to make approximate predictions about the time that a task will take and the final agent states.

Complementing the short-term planning with a learnt value function to form a hybrid planning/learning approach clearly offers a major additional advantage, and the difference is significant after 5000 trials (paired t -test, $p = 0.03$). The agents can (for example) make rational decisions about whether it is better to move fast to complete a task sooner, or to preserve energy. This is certainly not possible in the greedy planning approach.

Figure 4 shows learning curves for the feature weights in the value function. We note that `time left` is a significant state feature: clearly, the expected return to the end of the trial depends strongly on the time remaining. However it should be noted that this feature cannot directly affect behaviour because it will have the same value when comparing two plans with the same planning horizon. Of the various features describing the energy status of the agents, `min energy` is the most significant. It has positive weight, indicating that situations with large minimum energy have higher value. The agents learn to take actions which maximise the energy of the one with least remaining, to balance the burden of tasks.

This provides evidence that the group as a whole is more effective than its individual components: if some agents “drop-out” before the end of the trial, the others have difficulty responding effectively to the incoming task stream. In contrast, `mean energy` has little impor-

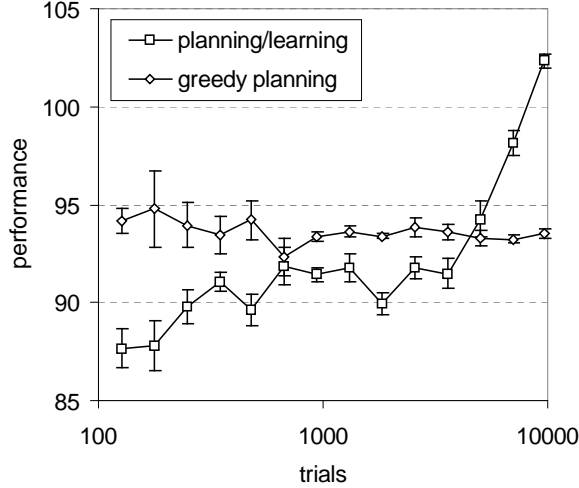


Figure 6: Performance comparison: non-endurance scenario.

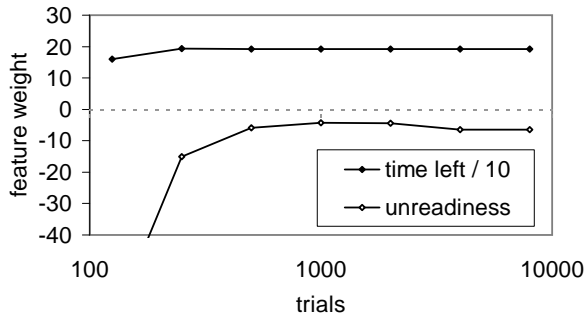


Figure 7: Learnt feature weights: non-endurance scenario.

tance. The remaining two energy features (max energy and median energy) have negative weight, but no obvious conclusions can be drawn because there may be a complex interaction with the min energy feature: removing one feature from the value function can often change the signs of lesser features.

A second scenario that does not require endurance was also studied. The only difference is that agents are given enough initial energy to complete each 10000-step trial at maximum speed, and so the energy features have very little part to play. The performance results (Figure 6) follow the same pattern as for the previous scenario: planning outperforms the simple heuristic, and learning increases performance further. (The heuristic performance remained below 50 and is not shown.) The benefit of learning is significant ($p < 0.01$) after 7000 trials, and does not level-off within the experiment duration (10000 trials).

The analysis of feature weights in Figure 7 shows that time remaining is the strongest feature. However, the positional feature *unreadiness*, which measures mean distances of closest agents to reference points, plays the most important role because it can affect behaviour. A large negative weight was learnt. This was expected because a high value of *unreadiness* causes agents to take longer to reach new tasks that appear within the operational area. There is scope for additional features to be added which encode more detailed information about the relative positioning of the agents. The remaining (energy) features have been omitted from this figure because they have little influence on learnt behaviour: negative weights are obtained because this encourages agents to select the maximum speed.

5 Discussion

These results have shown that a hybrid planning/learning method offers major advantages over simple heuristic rules such as assigning the closest agent to each task. In the scenarios given, performance has been approximately doubled. The performance improvement will depend strongly on the nature of the task stream. If there are few tasks entering the system, every task can be completed successfully using a naive policy. If there are many tasks, the need to limit energy consumption through speed control and effective positioning becomes important. Furthermore, if there are tight time constraints on individual tasks, it is likely that agent positioning will become very important, and have a major impact on performance.

The experimental evaluation highlighted the need to learn different value functions for different classes of problem instance. For “endurance” problem instances, where the aim is to operate the group of agents for as long as possible, the value function will be sensitive to the preservation of resources (*e.g.* energy). In contrast, for “rapid response” problem instances that require tasks

to be completed within tight time constraints, *positioning* features will be the major influence in the learnt value function.

6 Conclusions

We have developed a hybrid planning/learning system that allows scheduling of a group of agents for a heterogeneous stream of tasks. It was assumed that the task stream would be unpredictable, preventing planning beyond a short time horizon. The risk of using short-term (“greedy”) planning is that tactical factors (positioning of agents in readiness for new tasks) and strategic factors (preserving resources) are ignored. Therefore reinforcement learning (exploiting a fast system simulation) was applied to obtain a value function for taking these factors into account.

A short-term planning element remained necessary because it is able to properly take account of the actual tasks that are currently known to the system. This short-term planning uses task performance models that are able to predict the end-state and reward for each task, given a agent assignment. Initial experiments indicated a benefit from both the short-term planning and the learning elements of the system, compared with a naive heuristic approach that assigns the closest available agent to each task.

Acknowledgements

This research was funded by the UK Ministry of Defence Corporate Research Programme in Energy, Guidance and Control.

References

- Richard E. Bellman. *Dynamic Programming*. Princeton University Press., 1957.
- Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.
- B. P. Gerkey and M. J. Mataric. A formal framework for study of task allocation in multi-robot systems. Technical Report CRES-03-13, University of Southern California, 2003.
- J. J. Martin. *Bayesian Decision problems and Markov Chains*. John Wiley, New York, 1967.
- Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Solving very large weakly coupled Markov decision processes. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98)*, pages 165–172, Menlo Park, July 26–30 1998. AAAI Press. ISBN 0-262-51098-7.
- Malcolm J. A. Strens and Andrew W. Moore. Policy search using paired comparisons. *Journal of Machine Learning Research*, 3:921–950, 2002.
- Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. MIT Press, Cambridge, MA, 1998.
- Christopher J. C. H. Watkins. *Models of Delayed Reinforcement Learning*. PhD thesis, Psychology Department, Cambridge University, Cambridge, United Kingdom, 1989.

Interactive Multi-agent Learning

Pedro Rafael Graça

*Department of Computer Science, University of Lisbon
Bloco C5 - Piso 1 - Campo Grande,
1700 Lisboa, Portugal
prafael@di.fc.ul.pt

Graça Gaspar

†Department of Computer Science, University of Lisbon
Bloco C5 - Piso 1 - Campo Grande,
1700 Lisboa, Portugal
gg@di.fc.ul.pt

Abstract

If agents are to learn as a team and benefit from each other's effort, single agent individual learning mechanisms must be extended towards a multi-agent learning perspective. To do so, additional ways of sharing and gathering information have to be considered so that the agents can interact during the learning process. Since a multi-agent system is a system of agent interaction, its inborn potential can be used to serve this purpose. In other words, rather than simply extending the traditional individual learning mechanisms with specific interaction protocols, the activity that concerns the intermediate steps of the collective learning process can use the structure of the multi-agent system, benefit from its potentialities, and follow the flow of agent interaction. This form of multi-agent learning is called interactive multi-agent learning and there are good reasons to expect that it can lead to better results in dealing with certain problems that appeal to collective learning.

1 Introduction

In this paper we address the recent field of study of interactive multi-agent learning (IML), proposing answers for two main questions:

- Why should interactivity be introduced in multi-agent learning?
- How can multi-agent learning become an interactive process?

Concerning the first question, we discuss the costs and benefits that can result from IML and compare it with other multi-agent learning perspectives. As for the second question, we discuss what kind of information should be exchanged between the agents and how should that exchange occur in an interactive way. Furthermore, we complement the theoretical discussion by showing how to modify and extend a case-based reasoning traditional individual learning mechanism so that it can be used as an IML mechanism.

IML is a field of study that, although considered interesting and promising, has not been broadly researched yet. Since its boundaries are still somehow undefined, we will begin by distinguishing it from other types of multi-agent learning. For that, we will consider the three categories proposed by Weiss and Dillenbourg (1999) and present our own generic definition of IML.

We will use an example that helps to illustrate these three categories. For this effect, imagine a place where meals can be bought through three different services: take away, snack bar or restaurant. Imagine also that this place, called "Three Orientals", serves traditional food from India, China and Japan, and that all the recipes used are

exclusive and secret. Moreover, these recipes are very difficult to learn and take a long time to master.

1.1 Multiplied learning

In this type of multi-agent learning, each agent has an individual learning mechanism and does not share his results. The learning effort is multiplied by every individual involved in it. Multiplied learning can be a solution when agents cannot share information (privacy, hostility) or when communication cannot be supported (communication too expensive, lack of resources).

As an example, consider that the "Three Orientals" has three isolated kitchens, one for each of the three different meal services. Imagine that three chefs are hired, one for each kitchen. Since they do not communicate with each other, each one of them has to learn how to cook all the secret recipes by himself. The learning effort is multiplied.

1.2 Divided learning

When information exchange is possible, multiplied learning becomes redundant. The division of the learning task among the agents reduces this redundancy, but depends on several initial definitions:

- Which are the learning subtasks;
- Which agents execute each subtask;
- When and how is the knowledge (acquired in each subtask) transmitted and integrated.

These initial settings are the limits within which divided learning occurs, but such static definitions may not be possible (for example, the learning task may be too complex to be clearly divided and distributed) or desirable

(for example, in an unstable or highly unpredictable environment, the transmission of information may become too difficult and that may hinder the sharing of acquired knowledge on preset occasions).

Consider now that the "Three Orientals" has only one kitchen. The learning effort can, for example, be divided among the three chefs in the following ways: one learns to prepare the Indian dishes, the second one the Chinese dishes and the third one the Japanese dishes; one learns how to prepare the ingredients, the second one how to cook and the third one how to gather the cooked food and arrange the dishes. By dividing the learning task, the redundancy of the collective effort is greatly reduced. Notice however that such initial divisions are not always adjusted. For example, if all the services of the "Three Orientals" were reserved by the "Sushi Lovers Association" during one month, neither of the suggested divisions would be appropriate, because during that time only one of the chefs would be actively learning.

1.3 Interactive learning

In interactive multi-agent learning there also exists a division of the learning task, but in this case the division is not initially set. The two basic principles of IML are:

- to let the succession of environmental events and social interactions guide the path of the learning process;
- to use the potentialities of multi-agent systems on behalf of the learning process.

The idea behind these principles is that, by considering the successive environmental and social states to dynamically define well adjusted learning sub-tasks, by selecting agents to execute these sub-tasks and ways of sharing knowledge, and by using the means of social interaction provided by a multi-agent system to benefit distributed learning, the global results of the collective learning process can be enhanced. In IML, learning ceases to be a process which is isolated from other social processes. Instead, as it happens in human societies, it assumes its place inside the social system and follows its patterns of interaction.

In the shared kitchen of the "Three Orientals", the process of interactively learning how to cook the secret recipes takes place in an unpredictable way. The three chefs communicate frequently with each other and decide what is the best thing to do at each moment. They influence each other's learning path with advices and opinions. With this system they can, for example, identify relationships between the ways of preparing the same ingredients for different dishes; for instance, part of the knowledge acquired by one chef while cooking Chinese rice can help another chef that is learning how to cook Indian rice. When the "Sushi Lovers Association" reserves the services of the "Three Orientals", the three chefs adjust to the situation, divide the available tasks among them and are always able to keep learning.

2 Related work

Weiss and Dillenbourg (1999) address the subject of multi-agent learning in a general way, and propose a distinction between multiplied, divided and interactive learning. To them, it is this last category that may "explain all the benefits of multi-agent learning". Kazakov and Kudenko (2001) consider interactive learning to describe "true distributed learning" and they add that "very little research exists in this area".

Some works (Makar et al. (2001) and Turner et al. (2002) are examples) address the subject of multi-agent reinforcement learning in ways that show some degree of agent interactivity.

Ontañón and Plaza (2002) introduce a case bartering protocol to improve multi-agent learning. Although their work addresses agent interaction over previously learned information, it does not discuss the exchange of data during the learning process.

Nunes and Oliveira (2003) discuss the benefits of mutual agent interaction during the learning process, and propose an advice exchange technique that allows cooperative learning. Although we also propose advice exchange (among other ways of interaction) in our investigation, there are important differences in the way the exchange is made and in the kind of information exchanged.

Graça and Gaspar (2003) show that, on a dynamic multi-agent environment, the collective learning task can be optimised through simple agent interaction.

3 Interactivity and communication

Learning in an interactive way depends on the existence of a communication system that allows information exchange so that the learning path can be successively defined and the partial results of the learning effort can be efficiently shared. The type of information exchanged and the ways in which this exchange occurs must express this interactivity.

3.1 Type of information exchanged

The interactive circulation of specific knowledge that is related to the learning effort constitutes the core of IML, allowing agents to share information and complement their individual experience with external data gathered from other agents. The agents can share not only raw results (for example, a set of cases in case-based reasoning) and refined results (for example, a generalization of a set of cases), but also specific units of information that allow a richer interaction during the learning process. These units of information can be such as: a problem description; a solution (for example, the solution that an agent would select to solve a given problem); a justification (raw or refined data that supports a proposed solution); an evaluation (a measure of how good a given solution is for solving a problem). Such units can be interactively

created by the agents (according to their current experience) and allow the exchange of intermediate results (hypothesis) of the learning process. Here are some examples of messages that can be used to regulate the transmission of this information and allow the exchange of partial results at any intermediate stage of the learning process:

- Suggestion: A possible solution for a problem.
- Opinion: An evaluation of how adjusted a possible solution is for a problem.
- Help request: A request for advice from another agent that includes a problem description. It may also include a suggestion and its justification.
- Advice: The answer to a help request that may include a suggestion and its justification.

As mentioned before, information that concerns the state of the environment and of the multi-agent system can also circulate in order to help the definition of sub-tasks and the selection of agents to execute them. Here are some ideas of how the flow of social interactions can help to define the path of the learning process:

- Learning sub-tasks: At each moment, the current state of the environment can be taken into account for the definition of sub-tasks that can be executed in an easier or more advantageous way;
- Selection of agents: The agents' individual state can be considered as a selection criteria (for example, the importance of the current task can be taken into consideration); the current relations between agents can be considered when forming teams to execute learning sub-tasks;
- Knowledge sharing: When in need, the agents can ask for the help of other agents; agents can form groups to debate different opinions on possible solutions for a problem; contacts established for other purposes can be used as a way of also sharing experience acquired through learning.

3.2 Ways of Information Exchange

The generic communication protocols that, in each multi-agent system, serve agent interaction, can be used to exchange information that regards the learning process. For instance, when agents communicate while executing a non-learning task, it may be possible to use this contact to also exchange learned data (for example, suggestions could be made). In order to make better use of agent interaction on behalf of the learning process, it may be useful to introduce specific interaction protocols. These protocols may facilitate the transmission and exchange of specific data units (for example, providing the diffusion of help requests) and support specific forms of group discussion that address cooperative learning (for example, voting schemes).

4 Costs and advantages of IML

Not all problems that involve multi-agent learning appeal for interactivity. For instance, when there are strong re-

strictions to communication, when the agents do not cooperate or are hostile, or when the problem has a degree of predictability according to which a sequence of collective learning steps (divided learning) can be easily set, then IML may not be a viable or advisable option. The environments that appeal for interactivity are those that concern multi-agent learning on dynamic and unpredictable problems that can be solved through mutual cooperation. It may be adjusted to say that, in these cases, it is through IML that the potentialities of a multi-agent system can better serve the learning process; however, since the introduction of interactivity in learning has relevant costs, it is essential to discuss the reasons that lead us to believe that IML mechanisms can allow better results.

4.1 Costs of IML

The introduction of interactivity in multi-agent learning has important costs that must be considered. First, the ways in which agent interaction occurs during the learning process have to be planned, and this plan is generally more complex than a static initial definition (as it happens in multiplied or divided learning). Second, communication resources must be available and allow frequent information exchange. Third, agents have to be able to analyse and assimilate the information that circulates interactively during the learning process.

4.2 Advantages of IML

Comparing interactive and divided learning, we can say that the former extends the latter (the division of the learning effort becomes interactive) or that the latter is a simplified form of the former (interactivity is predetermined). Regarding this, we discuss several advantages of IML using divided learning as a reference.

According to the divided learning perspective, results can only circulate after the learning sub-tasks are completed. When this circulation occurs at any intermediate steps of the learning process (IML perspective), useful information can be assimilated during the performance of sub-tasks, leading to the decrease of redundancy in the collective effort and allowing interactive adjustments to the path of current sub-tasks. This can also help to avoid excessive specialization (that can result from undertaking specific sub-tasks without intermediate interaction), and create a diversity of perspectives that allows different ways of addressing the same problems and the exploration of new solutions.

When the learning effort is divided, there exists an initial notion of which agents will execute which learning sub-tasks, and when will they execute them. Specially on dynamic and unpredictable environments, considering that specific conditions may temporarily hinder the execution of some of these sub-tasks, such initial settings may become misadjusted. In these cases, the possibility of interactively defining sub-tasks and agents to execute them

can allow selections that match the current conditions.

The division of the learning task creates dependencies between the agents. If an agent fails to execute his task, his fault can seriously delay or even disrupt the collective effort. In order to avoid this, the accomplishment of each critical duty must be assured. This may prevent the application of divided learning mechanisms to environments where agents are to have a considerable degree of autonomy and be able to decide to stop learning during a period of time (for example, to attend to more important immediate tasks) or stop collaborating (for example, to attend to a new privacy policy), or in which faults in the communication system may occur and cause the isolation of agents for a significant time extent. The flexibility of IML allows the collective learning task to proceed in these cases. If necessary, the learning effort may temporarily assume a multiplied perspective (for example, when communication becomes unavailable) and reassume its interactivity when possible.

As a reference, imagine the cooperation between a group of ecologists and a group of travel agents over the management of a natural reserve. The ecologists wish to preserve the reserve but need money to take good care of it, and the travel agents want money but they need a well-kept reserve to attract tourists. In learning problems that appeal for this kind of cooperation, IML mechanisms can be specially well-adjusted, allowing the crucial interactive exchange of information between sets of agents that perform different but complementary tasks.

Attending to these potential advantages, there are good reasons to broaden the research of IML, for it may allow a faster collective learning with better results, provide better tolerance to agent or system faults, and address a wider scope of problems.

5 Cooperative case-based reasoning

At this stage of our research, we are focusing our attention on the adaptation of a case-based reasoning (CBR) traditional individual learning mechanism so that it can be used as an IML mechanism. We propose a cooperative CBR generic learning system that allows agent interaction during the collective learning effort. In doing this, we wish to illustrate how some of the ideas presented above can be applied.

5.1 Traditional and Multi-agent CBR

The traditional individual CBR generic learning process is based on the following four basic steps:

1. Retrieval of previously gathered cases (that concern problems somehow similar to the current problem);
2. Selection of a solution;
3. Revision of the solution (resolution of the problem and evaluation of the solution);
4. Storage of a new case.

In multi-agent CBR, this generic individual process has to be modified in order to include ways of information exchange between the agents. The information used in the traditional CBR process is retained in a single case base, whose contents express, at each moment, the experience acquired. In multi-agent CBR, different types of case bases can be used: individual or collective, centralised or distributed. Our research focuses on the use of individual case bases, a scenario in which information exchange is crucial for the collective learning process, allowing in this sense a more deep and complete discussion concerning the introduction of interactivity. The use of separate case bases can be justified for reasons like the need for privacy (for example, the existence of a secrecy policy concerning some of the data retained in individual case bases) or the need for efficiency on data access or storage (for example, to avoid traffic bottleneck situations that can result from having a collective case base).

5.2 Cooperative CBR

Using the traditional mechanism as reference, we have identified three situations that are suitable for interactive information exchange. The first is during the process of consulting previously gathered cases and deciding which solution to select to solve the current problem: instead of simply consulting his individual case base, an agent may request the help of other agents, consider their advices and discuss possible solutions with them. The second situation is during the revision of the chosen solution: while trying to solve a problem, an agent may again consult other agents to exchange information and interactively adjust his procedure. The third situation is during the storage of a new case: the new case can be shared among several agents (possibly by those involved in the previous two situations). In Table 1, we extend the basic CBR steps in order to incorporate these information exchange substeps.

Depending on each agent's individual desire, the opportunities for interactive exchange of information can be used or ignored. The information that circulates whenever an agent decides to ask for help can be useful not only to himself but also to the other agents contacted. For instance, during the process of selecting a solution for a problem, if the analysis of the information is done in group, then all the agents involved have the opportunity to retain useful knowledge that circulates during the discussion.

5.3 Agent interaction in cooperative CBR

The interactive information exchange between the learning agents involved in cooperative CBR appeals for specific processes of interaction. Considering the first of the three situations of interactive information exchange identified before, we propose two different protocols to regulate the process of accessing external information,

Table 1: Cooperative CBR learning process

Basic process	Extensions
I. Information retrieval	a) Retrieval of individual cases
	b) Access to external information
II. Selection of a solution	a) Analysis of internal and external information
	b) Selection of a solution
III. Revision of the solution	a) Access to external information to adjust the solution while solving the problem
	b) Evaluation of the solution
IV. Storage of a new case	a) Creation of a new case
	b) Distribution of the new case for storage in different individual case bases

analysing it and selecting a solution (steps I.b), II.a) and II.b) of the cooperative CBR learning process presented in Table 1). In these two protocols, the analysis of the information and the decision is centralised on one agent. This makes the process simpler and less expensive (regarding resource consumption); the distributed variants (ways of group discussion, voting schemes) are more complex and expensive, but offer powerful options for interaction. Choosing between centralised and distributed interaction depends upon the specific nature of the learning problem and the available resources. In the tables ahead (tables 2 and 3) we show the details of the two protocols, specifying how each step corresponds to the substeps of the generic cooperative CBR learning process.

Let A be the agent that is searching for a solution and that decides to ask a group $G = \{B_1, B_2, \dots, B_n\}$ of n other agents for advice. In the first sequence (Table 2), agent A sends the same help request to all agents of G , gathers their advices, analyses the information available, and then selects a solution.

In the second proposed sequence (Table 3), agent A sends one help request at a time, and waits for each answer. In this case, agent A changes the contents of his help request whenever he receives a better suggestion, one that has a justification that is considered stronger. Considering that justifications are composed by sets of cases or its generalizations, a stronger justification may, for example, be one that: contains the case that better matches the current problem and/or has a better solution (according to the evaluation criteria in use); expresses a higher experience level of the agent proposing it.

It is important to notice that the process of selecting a solution can consist in more than simply choosing one of the suggestions; instead, the information received can be used to compose solutions that combine aspects of different suggestions. These sequences describe the process of interaction from the point of view of agent A . There

Table 2: Information exchange: fixed help request

Step 1 - Help request generation
Agent A consults his case base and generates a help request $Hreq = [Problem, Suggestion, Justification]$ (corresponds to I.a)).
Step 2 - Information exchange
2a) Agent A sends $Hreq$ to each agent of G (corresponds to the first part of I.b)).
2b) Each agent B_i of G answers by sending his advice $Adv_i = [Suggestion_i + Justification_i]$ to A (corresponds to the second part of I.b)).
Step 3 - Selection of a solution
Agent A analyses (corresponds to II.a)) the information available and selects a solution (corresponds to II.b)).

Table 3: Information exchange: iterative help request

Step 1 - Help request generation
Agent A consults his case base and generates a help request $Hreq = [Problem, Suggestion, Justification]$ (corresponds to I.a)).
Step 2 - Information exchange
Agent A exchanges and analyses information (corresponds to I.b) and II.a)) according to the following algorithm: -Initialise $Hvar = Hreq$ -For each agent B_i of G do: a) Send $Hvar$ to B_i b) Receive $Adv_i = [Suggestion_i + Justification_i]$ c) If $Justification_i$ is stronger than $Justification$, then: $Suggestion = Suggestion_i$, $Justification = Justification_i$
Step 3 - Selection of a solution
Agent A analyses (corresponds to II.a)) the information available and selects a solution (corresponds to II.b)).

is however another process involved in this interaction: the process of generating an advice that is performed by each agent B_i of G . In Table 4 we present a generic sequence of steps that describes this process. This process also involves information retrieval and the selection of a solution, and in this sense, some of its steps also correspond to some of the substeps of the generic cooperative CBR learning process.

The second situation of possible agent interaction is during the resolution of a problem. At this time, an agent may again consult other agents to exchange information and interactively adjust his procedure. The processes that regulate this interaction are similar to the ones proposed

Table 4: Process of advice generation

Step 1 - Information retrieval
1a) Agent B_i receives a help request $Hreq$ from agent A ($Hreq = [Problem, Suggestion, Justification]$).
1b) B_i consults his case base and gathers information on how to solve the current $Problem$ (corresponds to I.a)).
Step 2 - Composition of a suggestion
Considering the retrieved information (if there is any), B_i composes a $Suggestion_i$ and its $Justification_i$.
Step 3 - Creation of an advice
If B_i considers his $Suggestion_i$ useful (after comparing it with the $Suggestion$ received from agent A and also considering both justifications; corresponds to II.a)), then he creates a new advice ($Adv_i = [Suggestion_i + Justification_i]$) and sends it to agent A (corresponds to II.b)).

for the first situation. The third situation occurs during the storage of a new case, when it can be shared among several agents. The process associated to this third situation consists in simply performing a selection of a group of agents to receive the information and send the new case to them. The crucial point in this process is the selection criteria. One idea is to consider those agents involved in the previous two situations and analyze the information they shared (their advices) in order to choose good candidates (for example, an agent whose advice shows lack of experience could be considered a good candidate).

5.4 Example of agent interaction

The following example resumes the application of the processes of agent interaction previously described. Furthermore, it suggests the kind of motivations and situations that may trigger the interaction and define how each agent gets involved in it. Suppose that after consulting his case base, agent A finds out that he has little experience on how to solve his current problem. Realizing that, he generates a help request (that includes a suggestion and its justification) and decides to use a variable help request protocol to send it to agents B , C and D . Agent B is the first to be contacted and answers that he has no experience in solving the current problem. Agent C is the second to receive A 's request and, after consulting his case base, he replies with an advice that suggests a different solution. Upon analysing C 's advice, A decides to modify his help request (replacing the current suggestion and its justification with the ones received from C) and then sends it to the third agent. Agent D analyses the request and concludes that he agrees with the current suggestion. In his advice he confirms the suggestion and adds his own justification. After receiving this last advice, agent A de-

cides to accept the external suggestion to solve the current problem. During the resolution of the problem, he finds no need for external help (he skips the second situation of interaction). After solving the problem, agent A creates a new case, adds it to his case base, and, since agents C and D already have considerable experience in the subject, sends only one copy of the case to agent B . It is important to notice that whenever the information that circulates during the interaction is considered useful it can be stored by the agents. For example, since agent B had no experience on the problem for which his help was requested, he could have considered useful to store the information present in the help request.

6 Conclusion

We have proposed a definition of IML, discussed its costs and advantages, and described ways for introducing interactivity in multi-agent learning. Using a generic CBR learning mechanism, we showed how the general principles of IML can be applied. Furthermore, we proposed specific processes to regulate agent interaction during the learning task and showed how the information can circulate in an interactive way.

References

- P. R. Graça and G. Gaspar. Using cognition and learning to improve agents' reactions. *Adaptive Agents and Multi-Agent Systems*, Springer(series LNAI 2636): 239–259, 2003.
- D. Kazakov and D. Kudenko. Machine learning and inductive logic programming for multi-agent systems. *Multi-Agent Systems and Applications*, Springer(series LNAI 2086):246–270, 2001.
- R. Makar, S. Mahadevan, and M. Ghavamzadeh. Hierarchical multi-agent reinforcement learning. *Proceedings of the 5th Conference on Autonomous Agents (Agents'01)*, 2001.
- L. Nunes and E. Oliveira. Cooperative learning using advice exchange. *Adaptive Agents and Multi-Agent Systems*, Springer(series LNAI 2636):33–48, 2003.
- S. Ontañón and E. Plaza. A bartering approach to improve multiagent learning. *Proceedings of the 1st Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, 2002.
- K. Turner, A. K. Agogino, and D. Wolpert. Learning sequences of actions in collectives of autonomous agents. *Proceedings of the 1st Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, 2002.
- G. Weiss and P. Dillenbourg. What is 'multi' in multi-agent learning? *Collaborative Learning*, Pergamon Press:64–80, 1999.

Using XCS to Build Adaptive Agents

Zahia Guessoum^{*†}

^{*}OASIS

Laboratoire d'Informatique de Paris 6 (LIP6)

Zahia.Guessoum@lip6.fr

Lilia Rejeb[†]

[†]LERI

Université de Reims

rejeb@poleia.lip6.fr

Olivier Sigaud^{*}

[‡]OASIS

Laboratoire d'Informatique de Paris 6 (LIP6)

Olivier.Sigaud@lip6.fr

Abstract

To deal with dynamic changes of their environment, agents need an adaptive mechanism. This paper proposes an integration of classifier-based framework (named XCS) and an agent-based framework (named DIMA). The result of this integration is an adaptive- agent framework. It has been applied to simulate economic models.

1 Introduction

Dynamic and complex systems, such as economic markets, are characterized by a large number of agents and a dynamic environment. To deal with the dynamic and unexpected variations of their environment, adaptive agents are very useful. Several learning-based multi-agent systems have therefore been realized (see Kazakov et al. (2001), Kudenko and Kazakov (2002) and Kazakov et al. (2003)). The proposed solutions can be classified in two categories:

- Single agent learning: Agents can learn independently of other agents. Every agent is thus a simple learning algorithm and its environment is often static.
- Multi-agent learning: Each agent is endowed with a learning algorithm to build a model of its environment. The latter includes other agents.

Most realized works in multi-agent learning deal with real-life applications. The proposed solutions are thus often *ad hoc*, they cannot be easily reused to build other real-life applications. So, we still need works on generic adaptive agent models. The purpose of this project is to propose a generic adaptive agent framework (named XCS-Agent). This framework is the result of the integration of an agent-based framework (named DIMA (Guessoum and Briot (1999))) and a Learning Classifier System (LCS) (named XCS (Wilson (1995))). The application of XCS-Agent to simulate economic models has allowed to highlight the advantages of the proposed framework and to underline the open problems (coding complex environments, exploration/exploitation, ...).

This paper is organized as follows: Section 1 presents the example, Section 2 describes the framework XCS-Agent, Section 3 studies the exploration/exploitation

problem, and Section 4 gives an overview of the realized experiments to validate XCS-based agents.

2 Example

The considered application is the simulation of an economic model. In this application, we consider a set of firms in competition with each other within a shared market. A firm is defined by the following main parameters:

- K, the amount of capital available,
- B, the R&D budget,
- the state variables (X vector) represent the different types of resources (funds, people, equipment, ...) owned by the firm,
- the Y variables represent the performances of the firm. They are directly influenced by the X vector,
- the strategy the firm follows to allocate its resources,
- the associated organizational form. An organizational form is an abstract entity that gathers a set of similar firms. Similar firms have similar behavior and similar structure (see Baum and Rao (1999) and Guessoum et al. (2003)).

Moreover, a firm is characterized by its decision process which aims to select the most suitable strategy in a given context. This context includes the internal parameters (K, B, X, ...) and the firm's perception of the other firms (their K, their B, their Y, ...). Several solutions may be used to represent this decision process such as inference engine, case-based reasoning and LCSs (see Holland et al. (2000)). However, the use of classifiers is more suitable to the dynamic and unexpected variations of economic markets.

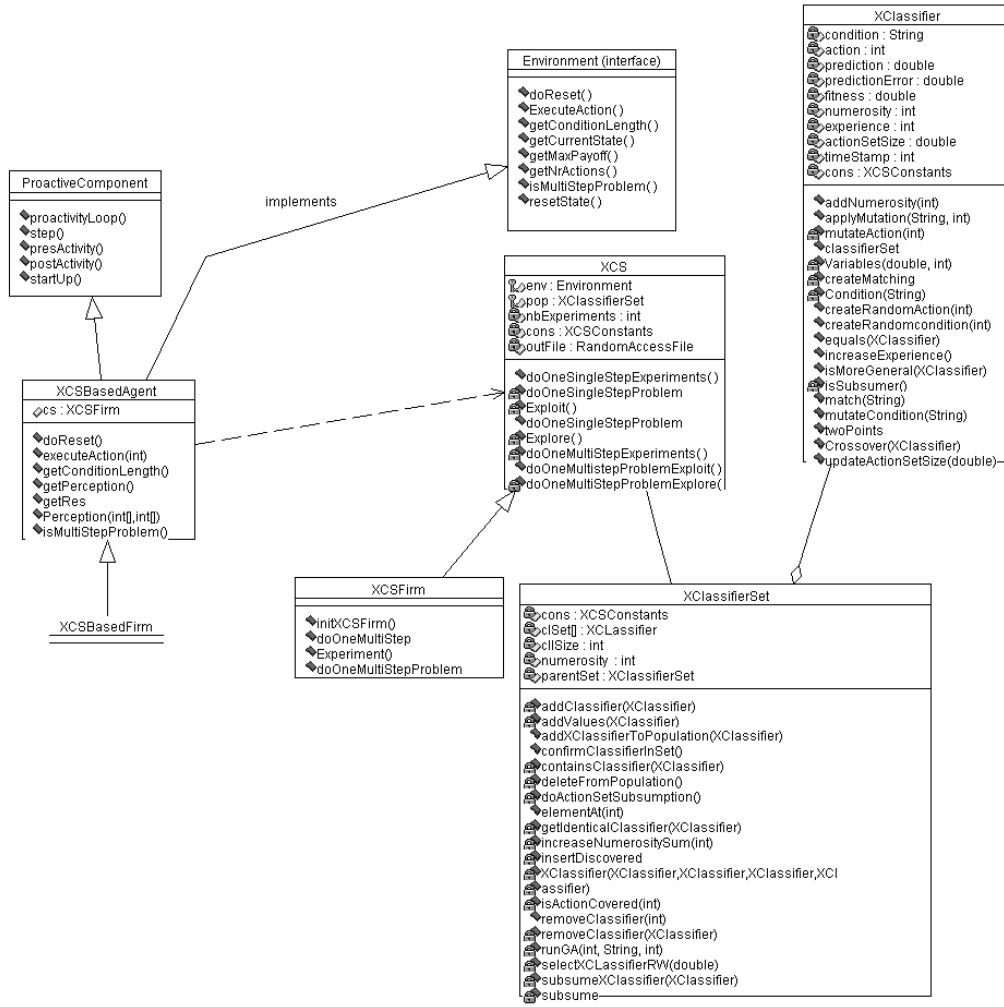


Figure 1: Overview of the framework XCS and XCS-Agent

3 XCS-Based Agent Model

This section presents first an overview of XCS, it then describes the XCS-Based agents.

3.1 Overview of XCS

XCS is a recent LCS (Wilson (1995)) which can solve complex learning problems. It is based on a standard classifier condition-action rules. Each classifier is characterized by three parameters: prediction p , error e , fitness f . A condition corresponds to a chain of 0, 1 and #. # represents 0 or 1, the associated attribute is not taken into account when checking the condition. The parameters p , e and f are automatically updated according to the reward obtained by the application of the chosen action.

In a context s , a step of XCS executes the following actions (see Table 1):

- scan the environment (define the state of the environment),

- execute a step by using the exploration or exploitation strategy.

XCS provides a set of generic classes which can be reused to implement LCSs (see Figure 1). To implement a LCS, one has to implement the Environment Interface.

XCS has been reused to build XCS-Agent. The agents context and their behavior are described in the following sections.

3.2 Agent Context

In a classifier, the condition represents the context of the agent. It is defined by its local parameters and its perception of the environment. For instance, the firm's context includes:

- the capital, the resources, the budget,
- a representation of the competition (information on the other firms),

Table 1: An example of method of the XCS step

```
private void doOneSingleStepProblemExplore (String state, int counter){
XClassifierSet matchSet= new XClassifierSet(state,pop,counter,env.getNrActions());
PredictionArray predictionArray= new PredictionArray(matchSet, env.getNrActions());
int actionWinner= predictionArray.randomActionWinner();
XClassifierSet actionSet = new XClassifierSet(matchSet, actionWinner);
double reward = env.executeAction(actionWinner);
actionSet.updateSet(0, reward);
actionSet.runGA(counter,state,env.getNrActions());}
```

- a representation of the organizational forms which is defined by the resource variations and the performances of the associated firms. Each variation of a resource is described by a symbolic value (small, medium, large). In our experiments, we use the same fuzzy granulation (Zadeh (2001)) for the various resources (see Guessoum et al. (2003)).

The various attributes of a firm are not binary. We have thus decomposed the definition domain of each attribute i in n intervals. An attribute can be thus coded by a binary string of n bits which indicate the corresponding interval. However, the decomposition of the definition domain into intervals is not easy and the performances of a firm rely on this decomposition. Indeed, if the interval boundaries do not fit with the natural boundaries of an optimal strategy, the adaptive agent cannot perform optimally (see Section 5).

An action corresponds to a strategy of the firm (see Section 2). An example of classifier is given in Table 2.

Table 2: Example of classifier

<i>condition</i> $K \in [-300, 100]$, $B \in [0, 100]$, $X[1] \in [2, 5], \dots, X[8] \in [1, 3]$ $AverY[1] \in [3, 20], \dots, AverY[3] \in [0, 3]$ <i>action</i> strategy1, parameters $P = 0.5$, $e = 0.01$, $F = 100$.

The capital intervals are: $[-300, 100]$, $[101, 300]$, $[301, 500]$, $[501, 600]$, $[601, 800]$, $[801, 1000000]$. In the condition of the given example (Table 2), it is represented by 7bits: 1000000. Each bit indicates if the value belongs to the corresponding interval.

The reward corresponds in our model to the aggregation of the variation of performances which result from the application of the chosen strategy. It is calculated by

the following formula:

$$r = agreg\left(\frac{Y_t[1] - Y_{t-1}[1]}{Y_t[1]}, \frac{Y_t[2] - Y_{t-1}[2]}{Y_t[2]}\right) \quad (1)$$

where *agreg* is an aggregation operator.

3.3 Agent Behavior

The used agent framework is DIMA (described in Guessoum and Briot (1999)). DIMA is a framework of proactive components representing autonomous and proactive entities. It is illustrated by a minimal set of classes and methods defining the main functionality of a proactive component. This functionality may be extended in the subclasses. This framework is mainly composed of the class ProactiveComponent (see Table 1) which describes:

- The goal of the proactive component, it is implicitly or explicitly described by the method `isAlive()`.
- the basic behaviors of the proactive component, a behavior is a sequence of actions that allow to change the internal state, to perform a message or to send a message to other proactive components. Each behavior is implemented as a java method of this class.
- the meta-behavior defines how the behaviors are selected, sequenced and activated.

The step of a firm is defined in Table 4:

Table 4: Step of an agent

<pre>public void step() { updateCompetitionRepresentation(); getProfitVariation(); updateBudget(); %% begin decision process budgetRest=applyStrategy(chooseStrategy()); %% end decision process updateCapital(); caculatePerformances(); updateMarket(); }</pre>

Table 3: Main methods of ProactiveComponent

Methods	Description
public abstract boolean isAlive()	Tests if the proactive component has not yet reached his goal.
public abstract void step()	represents a cycle of the meta-behavior of the proactive component.
void proactivityLoop()	Represents the meta-behavior of the proactive component. <pre>public void proactivityLoop() { while (this.isAlive()) { this.preActivity(); this.step(); this.postActivity(); }}</pre>
public void startUp()	Initialize and activate the meta-behavior. <pre>public void startUp() { this.proactivityInitialize(); this.proactivityLoop(); this.proactivityTerminate(); }</pre>

XCSBasedAgent (see Figure 1) is defined as subclass of ProactiveComponent and implements Environment. An XCS is associated to each XCSBasedAgent and its meta-behavior uses the XCS step (methods doOneMultiStep*). The step of an adaptive firm is defined in Table 5.

Table 5: Step of an adaptive firm

```
public void step() {
updateCompetitionRepresentation();
getProfitVariation();
updateBudget();
%% start decision process
cs.doOneMultiStepExperiment(3);
%% end decision process
updateCapital();
caculatePerformances();
updateMarket(); }
```

ration/exploitation rules to the evolution of the context and the state of the classifier set according to the variations of the firm performances. These meta-rules are mainly based on two parameters:

- m: the number of steps during which an agent uses exploration,
- n: the number of steps during which an agent uses exploitation,

After each m exploration steps, the system executes n exploitation steps. It executes then these meta-rules:

- if the $Perf(t+n) \leq Perf(t)$ then the system must still learn, the number of exploitation steps is then decreased ($n = n/2$)
- if the $Perf(t+n) > Perf(t)$ then the system has learned enough, the number of exploitation steps is then increased ($n = n*2$).

They are simple and adapt the behavior of the LCSs to the evolution of the agent environment. They provide thus a good solution to the Exploration/Exploitation dilemma.

4 Exploration/Exploitation

LCSs must find a good compromise between two complementary strategies: exploration and exploitation. When the uncertainty in the current prediction is high, the system should better explore than exploit (see Wilson (1996)). An adaptive agent should be able to observe its behavior and choose the most suitable strategy according to its experiences. To deal with that problem, we introduce meta-rules which allow to adapt the explo-

5 Experiments

The proposed framework was tested on the simulation of economic models (see Section 2). We first compared the XCS-Based firms and firms that use *a priori* defined rule-based systems. We considered two populations of firms: rule-based firms and classifier-based firms. We injected in each population one XCS-based firm and we observed the performances and the number of classifier of this firm. The considered parameters are:

- A population size =800
- A #_probability = 0.5
- a learning rate (b)=0.2
- a crossover Rate=0.8
- mutation rate = 0.02
- qGA =25
- minimum error = 0.01

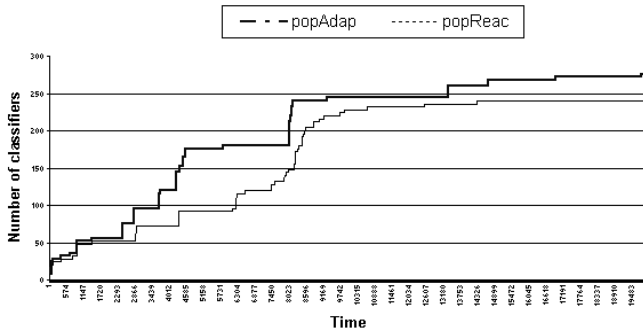


Figure 2: Convergence of the number of classifiers

The experiments show that the convergence of the classifier number within a population of non adaptive firms is easier (see Figure 2). In fact, in adaptive-firm population, the firms need a lot of time to learn and construct their classifier populations. These results are, nevertheless, sensitive to some initial values of the parameters of the XCS such as the learning rate. The learning coefficient beta is important in LCSs. Its default value, in XCS, is 0.2. To show the influence of this parameter, we realized experiments with different learning rates. Figure 3 shows that the reduction of this value improves the convergence.

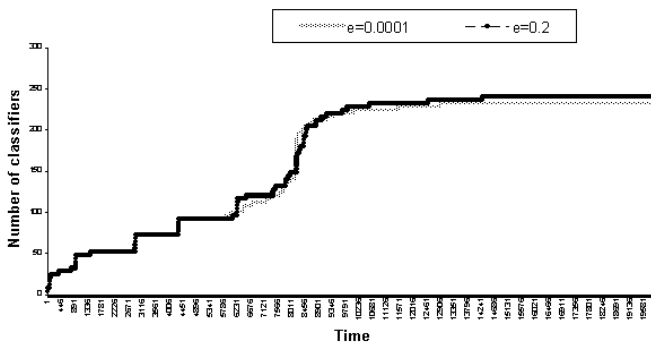


Figure 3: Comparison of the convergence of adaptive firms using different learning coefficients

We set then this learning rate to 0.0001 for the rest of the experiments. We study also the effect of the representation on the convergence of the firms. We use for this

two populations: 1) in the first population, the classifier representation is based on a representation on 8 intervals and 2) in the second one, the classifier representation is based on 16 intervals. Figure 4 shows that the more precise representation allows easier convergence. So in the rest of these experiments, we use 16 intervals.

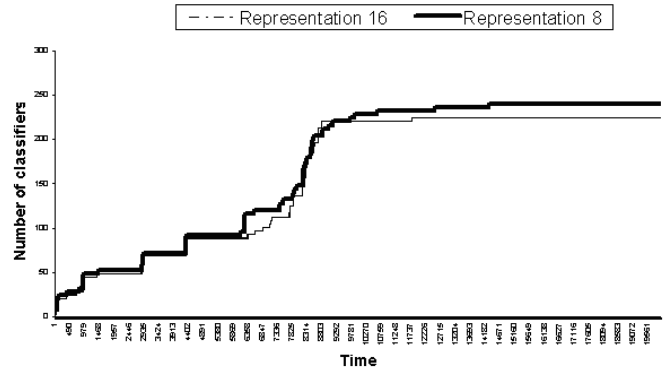


Figure 4: Comparison of the convergence of populations with different representations

In the second series of experiments, we studied the Exploration/Exploitation problem. We considered three populations of 500 XCS-based firms with different strategies: exploration, exploitation, and meta-rules (Explore-Exploit).

The results (see Figure 5) show that the population with meta-rules (the black one in the Figure) has slightly higher performances. We note that the difference is not important (5 %). These meta-rules are then a good technique but more experiments are needed to find the adequate parameters such as m.

6 Conclusion

This paper presented a new XCS-based agent framework and its application to simulate economic models. The latter are dynamic and complex systems. This application showed the advantages of using LCSs in dynamic multi-agent environments. Large-scale multi-agent systems provide thus very good applications to validate LCSs, but also very challenging ones, given the continuous and non-stationary character of these applications. The first experiments are interesting but more experiments are needed to choose the most suitable parameters and intervals to improve the performances. On that point, using adaptive interval techniques such as the one suggesting comparison of the profit of populations with different strategies (see Wilson (2000)) is a major area for future work. A second perspective of this work is the definition of a methodology to facilitate the development of adaptive-agent systems.

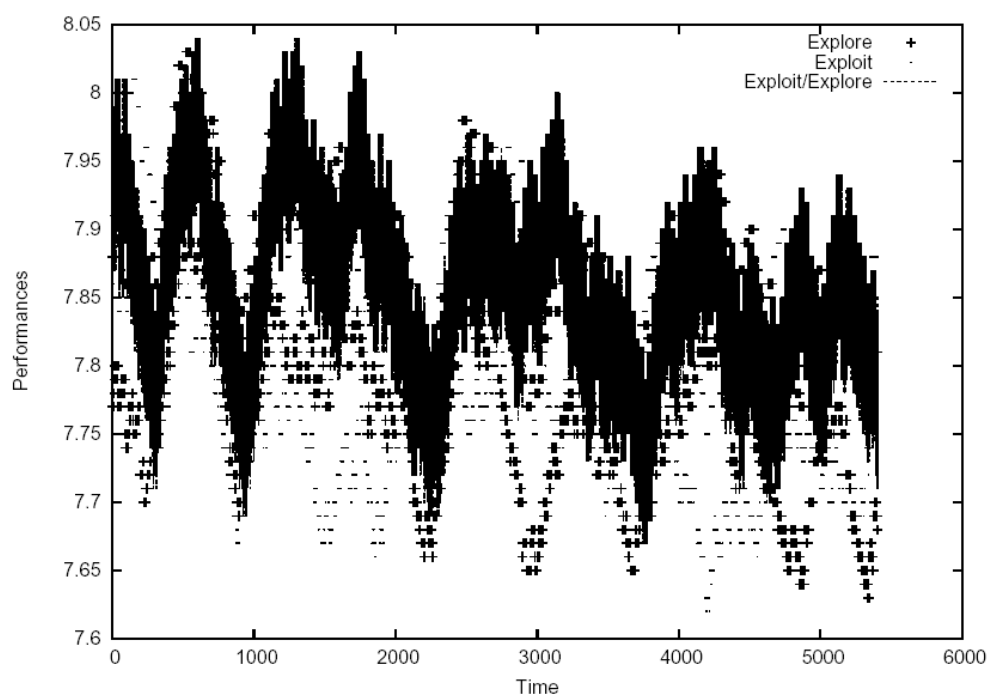


Figure 5: Comparison of the performances of the different populations with different strategies

References

- A. Joel A.C. Baum and Hayagreeva Rao. *Handbook of Organizational Change and Development*, chapter Evolutionary Dynamics of Organizational Populations and Communities. Oxford University Press, 1999.
- Z. Guessoum and J.-P. Briot. From active objects to autonomous agents. *IEEE Concurrency*, 7(3):68–76, 1999.
- Z. Guessoum, L. Rejeb, and R. Durand. Emergence of organizational forms. In *AAMAS'03*, Aberystwyth, UK, April 2003. AISB.
- J. Holland, L. B. Booker, M. Colombetti, M. Dorigo, D. E. Goldberg, S. Forrest, R. Riolo, R. E. Smith, P.-L. Lanzi, W. Soltzmann, and S. W. Wilson. What is a learning classifier system? in learning classifier systems: from foundations to applications. In P.-L. Lanzi, W. Soltzmann, and S. W. Wilson, editors, *Learning Classifier Systems: from Foundations to Applications*, pages 3–32. Springer-Verlag, Heidelberg, 2000.
- Dimitar Kazakov, Eduardo Alonso, and Daniel Kudenko, editors. *Adaptive Agents and Multi-Agent Systems*, York, UK, 2001. AISB.
- Dimitar Kazakov, Eduardo Alonso, and Daniel Kudenko, editors. *Adaptive Agents and Multi-Agent Systems*, Aberystwyth, 2003. AISB.
- Daniel Kudenko and Dimitar Kazakov, editors. *Adaptive Agents and Multi-Agent Systems*, London, 2002. AISB.
- S. W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
- S. W. Wilson. Explore/exploit strategies in autonomy. In J. Pollac J.-A. Meyer P. Maes, M. Mataric and S. Wilson, editors, *From Animals to Animats 4, Proc. of the 4th International Conference of Adaptive Behavior*. MA., Cambridge, 1996.
- S. W. Wilson. Get real! XCS with continuous valued inputs. in learning classifier systems: from foundations to applications. In P.-L. Lanzi, W. Soltzmann, and S. W. Wilson, editors, *Learning Classifier Systems: from Foundations to Applications*, pages 209–220. Springer-Verlag, Heidelberg, 2000.
- L. A. Zadeh. A new direction in ai: Toward a computational theory of perceptions. *AI Magazine*, 22(1):73–84, 2001.

The Strategic Control of an Ant-Based Routing System using Neural Net Q-Learning Agents

David Legge; Peter Baxendale
Centre for Telecommunication Networks,
School of Engineering,
University of Durham

d.n.legge@durham.ac.uk; peter.baxendale@durham.ac.uk

Abstract

Agents have been employed to improve the performance of an Ant-Based Routing System on a communications network. The Agents use Neural Net based Q-Learning approach to adapt their strategy according to conditions and learn autonomously. They are able to manipulate parameters that affect the behaviour of the Ant-System. The Ant-System is able to find the optimum routing configuration with static traffic conditions. However, under fast-changing dynamic conditions, such as congestion, the system is slow to react; due to the inertia built up by the best routes. The Agents reduce this drag by changing the speed of response of the Ant-System. For best results, the Agents must cooperate by forming an implicit society across the network.

1 Introduction

Ant-Based Routing has some attractive properties for communication networks due to the distributed nature of the algorithm. Schoonderwoerd [1996] and Dorigo [1997] described the first systems, which tackle slightly different problems (telephone networks and datagram networks). There have been numerous improvements described, but all have concentrated on the algorithms controlling the behaviour of the ants.

The drawback to Ant-Systems is that they are purely reactive in nature and the probabilistic advantage built up by the best route in good conditions becomes a disadvantage if that route becomes congested.

By using ideas from the area of Subsumption Architecture [Brooks 1986] – i.e. blending purely reactive agents with strategic proactive agents to create a more balanced society – this problem could be overcome.

The Ant-Colony on each node can be viewed as an agent - the ants being only messages, rather than each ant being viewed as an agent, which is an alternative model. It is then a logical step to place a second agent on a node to perform the more strategic role.

The Agent (as distinct from the Ant-Colony), can then monitor the operation of the Ant-Colony and adjust the parameters that affect the speed of response of the Ant-System. This should be done in a way that does not increase the level of Ant-traffic, since this will add to the congestion being tackled.

The Agents across the network must coordinate to some degree, since the affect of only one taking any action would be minimal. This would ideally be done without explicit coordination since the messages could also add to congestion; thus a parallel is drawn to Blind Game-Playing Agents studied in Reinforcement Learning, including [Kapetanakis et al. 2003].

2 Aim

The aim of this research is to show that an intelligent agent can be used to modify the parameters associated with the Ant-system to make both the convergence from start-up, and the response to changing conditions within the network, quicker, whilst retaining the stability in steady-state.

A society of Agents is required however, since the parameters are changed on a per-node basis for distributed control - an agent on each node is in control of the parameters used on that node. At this stage, the agents are acting independently from each other. Conversely, both the ant system and the agents use stigmergy – where the environment is the media for an implicit form of communication.

An agent *could* observe how other agents are acting by checking the parameters carried by ants originating from other nodes.

3 Current Work

A significant amount of research has investigated the self-organisational properties of Ant-Based systems within telecommunications – for both load-balancing and routing purposes.

Dorigo [1997], and many others, use two types of ants, one to explore the network, and a second type to propagate the information back across the network.

Other techniques include Vittori [2001], in which the probabilities are interpreted as Q-values, and the Q-Learning reinforcement equation is used to update them.

However, both these papers use single-valued parameters, such as rate of production of ants, rate of deposition of pheromones etc., that are empirically obtained. These are not changed at runtime, or indeed

off-line.

The Vittori paper utilises Q-learning to update the routing tables and found performance gains in doing so. In this paper, it is intended to use Q-learning to modify the parameters used by the ants.

[Kapetanakis et al 2002, 2004] discusses agents participating in games with other agents, where the reward gained at the end of the round is a function of the decisions made by all the agents. e.g. Agent 1 decides on action A, while Agent 2 opts for action B. The games are taken further by the addition of a probabilistic affect, so that the reward is $f(A, B, P)$.

Kapetanakis' agent's decisions are made without conferring with other agent(s), indeed the agent is not aware of the presence of other agents. The agents use Q-learning, and are able to find successful joint strategies. This is analogous to the situation described in this paper.

Other research effort has concentrated on improving and fine-tuning the rules that govern the behaviour of the ants. This paper attempts to branch away from these efforts and strategically manipulate the Ant-System in a similar way to Subsumption Architecture.

4 Ant-System

This section describes the implementation of the Ant-System used. It is broadly based on the system discussed in Schoonderwoerd [1996].

The basis for the ant system is many simple-acting entities displaying complex behaviour on the macro-scale. To this end the ants are small packets of data that are kept as simple as possible. Carried within the ant are: *Source*, *Destination*, *Amount of Pheromone* and *Deposition Rate*. Rather than carrying code with every ant, the processing is performed by the node.

The *Rate of Ant Production* is a self explanatory parameter, and at generation the ant is given a uniformly-random destination, chosen from all the known nodes on the network. It is also branded with an *Initial Amount of Pheromone* and a *Deposition Rate* (which describe its longevity and its potency). Once generated, and at each node along its journey it selects its next-hop by the generation of a uniformly random number and the consultation of the local probability table associated with the ants final destination.

On arrival at a node, and after ensuring the ant has not returned to its source, the ant deposits some pheromone. The amount of pheromone left by the ant is a simple percentage (defined by the Deposition Rate) of the current amount the ant has. If the ant has travelled in a loop, it is discarded.

Before this is added to the probability table (or pheromone table) it is multiplied by the (relative) available size of the traffic queue for the link just used by the ant.

The resultant amount of pheromone is added to the local table regarding the ant's source, i.e. The ant reinforces the return route, not the route it is currently taking. The tables are then normalised to sum to unity; this has the affect of rewarding the route just used, but also punishing all the routes *not* just used; a Minimum Probability is applied to ensure all routes are explored periodically. If the ant is at its destination it is treated

as above, then discarded.

The output of the ant-colony is the best next-hop for each known node. This is the highest probability link, except for when an extra threshold is applied – i.e. a link must exceed the probability of the current best route by some value for the recommendation to change.

The affect of the ant parameters on the probabilities have been investigated with and without traffic. It has been shown that it is possible to have a very quick response, but with a relatively noisy steady-state condition, or a smoother steady-state but a slower response. The Rate of Production also has an affect on the response.

An assumption is made that each and every link is bi-directional; the discussion of the validity of this assumption is beyond the scope of this paper. In Vittori [2001] routes are reinforced in both forward and backwards directions; although this may aid speed of convergence, the forward reinforcement has no logical basis, and so the feedback may be incorrect, leading to problems with incorrect routes.

The Ant-System is a separate entity, it is reactive in nature and can operate with or without the Agent. The Agent provides a supervisory and proactive role and is described next.

5 The Agent

An agent sits on each node and monitors the ant colony on that node. The agent receives reports from the colony, both regular ones and ones triggered by any route changes that occur.

The Agent is informed of all the parameters associated with the ants (Rate of Production, and those carried by the ant), as well as the threshold value for route change.

The agent is also told the current probability of the preferred route, as well as its mean, standard deviation and gradient. Obviously, the windowing function for calculating the mean, etc. is an important property and this has been tuned to highlight important features.

The success or failure of the agent can be determined by a comparison between the performance of the Ant-system by itself, and the Ant-system when supervised by the Agent.

The type of learning strategy employed by the agent is discussed in the next section.

6 Agent Learning Strategies

The agent is being asked to react to different states, and should be capable of learning the correct action from any state. The environment is also non-deterministic, so simple Temporal Difference Learning (TD-Learning) would not be suitable; therefore the preferred strategy would be Q-Learning - where an agent learns state-action values.

Although Q-Learning was originally based on tables of state-action pairs, this technique becomes unwieldy when there are more than a handful of states and/or actions. Other problems occur when the system is continuous and must be approximated to a specific state before an action is chosen.

This process of function approximation and the subsequent choice of action has been performed in a single step by Tesauro [2002] and applied to the game of Backgammon with remarkable success.

In the context of Backgammon, TD-Learning is sufficient, since the environment is predictable (the “after-position” of a move is deterministic). The extension of this approach to Q-Learning is a logical next step, and has been investigated by Tesauro [1999] himself.

Tesauro's approach uses an artificial Neural Network, NN, to approximate the Q-value for each action presented to the network for a particular state. Standard NNs are trained to perform known functions - e.g. the Exclusive-Or Function – where the correct inputs and outputs are defined. The error across the outputs can be calculated easily by taking the difference between the desired results and the actual results. The NN weights can then be adjusted accordingly.

Of course, with NN-based Q-Learning the correct outputs are the unknown; but if the Q-Learning Update Rule [Sutton & Barto 1998] is applied instead of the simple error, then the affect is directly equivalent to incrementing table-entries.

The NN-Based Q-Learning system can therefore be trained in the same way as a table-based system. Results from [Tesauro 1999] show that although the Q-values are shown to be less accurate than the table-based equivalent, the actions selected tend to be sound.

7 Implementation

The System has been implemented. The greatest challenge is to devise a Reward Function that adequately encourages desired outcomes whilst punishing undesirable results.

In the current configuration the Agent is able to adjust the *Initial Amount of Pheromone* the ants have – if the *Rate of Production* were to be adapted there is a danger that congestion would be caused by the control system attempting to reduce it.

The Agent therefore (usually) has three actions available to it – increase or decrease the *Initial Pheromone* or do nothing. The parameter is bounded as a precautionary “sanity-check”, in which case an action that would break this constraint is not presented as an option.

The representation of the actions to the NN is of note.

This can either be achieved by using a single input, and using arbitrary values to represent each value (e.g. -1 for decrease, 0 for stay the same, and +1 for increase) or, using three different inputs with a “1” input to show selection (0 otherwise).

The NN must have at least a single hidden layer to be able to approximate non-linear functions. Initially for speed of convergence and simplicity, only the single extra layer will be used. The output layer is a single node – the Q-value for the action considered – whilst the input layer is composed of the data inputs and the three action inputs. The hidden layer is composed of three nodes to match the action nodes of the input layer.

In order to achieve the objectives, the system must be allowed to train for a sufficiently long time; it is important to note that there is no hard distinction between training and runtime – the system will be constantly learning so that it is able to adapt to conditions. There could be a danger of “over-training”; however, given the noisy nature of the environment this is not anticipated to be a problem.

Different reward functions have been tested to find the most effective. Ideally, the reward would be composed of a smooth function of one of the (non-action) inputs – this negates the requirement to introduce an arbitrary threshold that may be suitable for some situations but not for others. This will allow the agent to adapt to the maximum number of conditions; it must be noted however, the the cost of generalising solutions is often prohibitive – in this case in terms of research time.

The two main statistical descriptions of the system that could be used as the reward function are the *standard deviation of the highest ant-probability* (referred to as *standard deviation* from now on) and the *magnitude of the change in the maximum probability*. These will be maximal when the conditions on the network are changing, so can be used as the detection mechanism.

8 Experimentation

The aim of the experimentation is to show that there is an improvement in performance when the parameters governing the ants are changed dynamically at run-time - thus taking account of changing conditions on the network.

Because of the inherent complexity involved in testing a network, two relatively simple simulations and then a more realistic network are used. These are described in the following sections.

Of interest in the test, is the reaction of the ant-system (both managed and un-managed) to congestion; the speed of response is of importance. A further concern is that the routing should not constantly change between two or more routes.

For both scenarios, the Managed Ants performance must be compared with that of the Unmanaged Ants. As intermediate steps, both a predetermined strategy, with *a priori* knowledge of the traffic conditions, and a simple reactive agent (using a simple threshold) are implemented – both should also the performance to some degree. This will give a further benchmark with which to contrast the performance of the full system.

8.1 The Simulator

The simulator being used is *ns* (Network Simulator) [Breslau 2000]. It is designed to model TCP/IP networks. These networks are connectionless; that is to say packets of data are transmitted without setting up a definite route first. This means that at each hop, packets are routed according to the instantaneous best route. Packets from the same flow do not necessarily follow the same path. Consideration of Circuit-Switched and Connection-Oriented networks are left to later discussion.

8.2 Network One

The network to be tested is shown in Figure 1(a). It is a simple five node network. The important feature is that there are three equally-long routes between Node 0 and Node 1, one of which, as described above, will at times be congested.

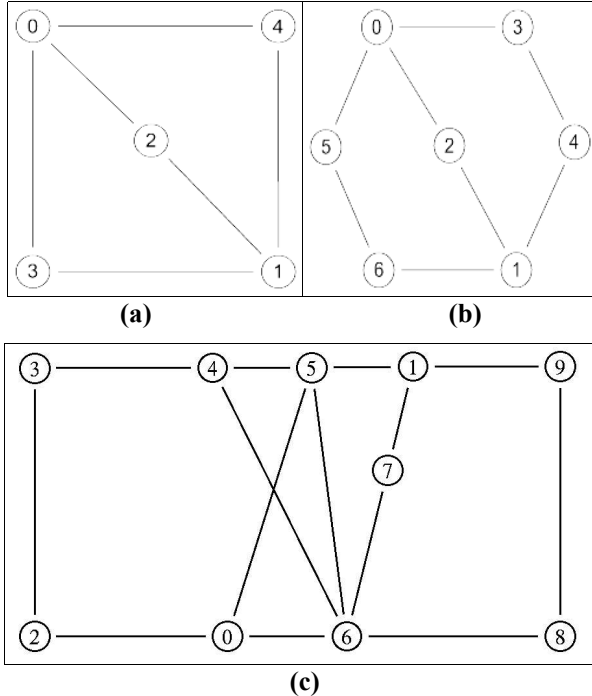


Figure 1: Diagrams of Networks One, Two & Three

8.3 Network Two

This scenario exists when there is a distinct shortest-path between two nodes as well as longer alternative routes, not sharing any links. The topology is shown in Figure 1(b).

Where the shortest route becomes congested, the ants must react as quickly as possible to the congestion. They must overcome a large amount of inertia represented by the high probability built-up by the shortest route - this inertia is greater than that of the previous scenario.

8.3 Network Three

The third network under test, Figure 1(c), is a more complex, but realistic, one. It is a sparsely connected network, with a number of cross-links. The route between Nodes 0 and 1 still has a number of alternatives.

8.4 The Traffic Scenario

Both network scenarios will be required to transmit some user-data, which we shall describe as real-time, loss-sensitive data. This shall be transported from Node 0 to Node 1, in all network topologies. The traffic shall be offered to the network intermittently over the whole simulation causing congestion numerous times.

Problems would be caused by the network dropping the user-data, excessive delay or excessive jitter - variation in the inter-arrival time of packets, often caused by switching of routes.

The situation under which the network will be tested is as follows. The traffic causing congestion, shall represent other network load and will be set at a level that will not, by themselves, cause packets to be dropped, other than ants.

The simulation will start with no data-traffic on the network. The user-data starts before the other sources. Firstly, a constant-bit-rate (CBR) flow uses the majority of the bandwidth available on the link between Node 0 and Node 2. Secondly, a smaller Variable-Bit-Rate (VBR) flow also uses the same link.

This combination of traffic will result in intermittent congestion during the time when both flows are present on the network. During congestion, traffic will require storage in queues waiting to be transmitted and ant-packets will be dropped, having been assigned a lower priority than other traffic.

These two flows both stop after a total of fifteen seconds. In the case of the Learning Agent, the congestion is repeated in an on-off fashion to allow the NN to learn the correct response.

8.5 Comparison Tests

The Agent-Managed Ants must be compared with the ordinary Ant-System to show that an improvement has been made. Therefore the basic Ant-System by itself is used as a control test.

Further comparisons could be made with both an *ideal* agent (that is not constrained by processing power/time) and a simple reactive agent (to confirm whether the overhead of the Learning mechanism is actually needed).

The simple reactive agent could just use a simple threshold mechanism, so when the standard deviation exceeds a pre-set level the age is increased, but is reduced when it falls below.

The expected shortcoming of this simple scheme is that it will undoubtedly cause excessive noise during steady state, and setting the threshold to the correct level is a non-trivial task as it may not be the same for every network. For instance if there are numerous links from one particular node then the standard deviation will be higher on average than on a node with few links.

9 Results

Results show that the Agent is able to improve the speed of reaction of the network to dynamic traffic scenarios; by increasing the Initial Age in congestion, but reducing it otherwise. Comparisons are made with the ordinary Ant-System to evaluate the effectiveness.

9.1 Network One

Figure 2 shows two overlaid graphs for Network One. The graph in the foreground is the probability plot for Node 0 to Node 1. The graph behind shows

the agent manipulating the Ant-System – the solid looking block shows that the agent has increased the parameter to increase the response-speed of the system. It can be seen that this occurs when there is a disturbance on the network. At the time of congestion, the congested route is not the selected one, so no quantitative results are relevant.

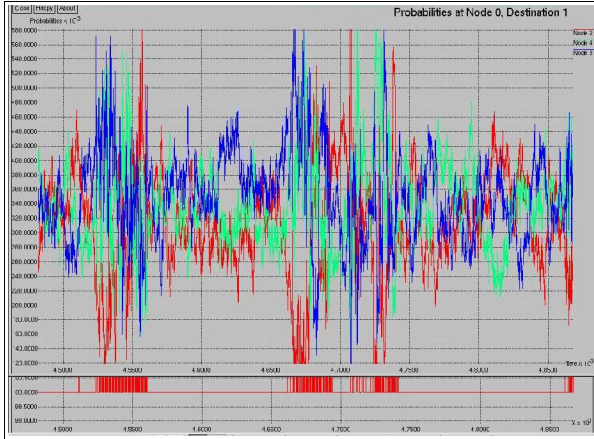


Figure 2: Graphs of probabilities and Agent manipulated Ant-parameters on Network One

9.2 Network Two

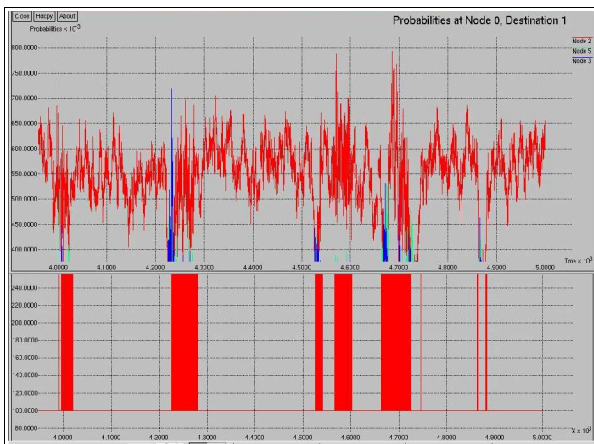


Figure 3: Graphs of probabilities and Agent manipulated Ant-parameters on Network Two

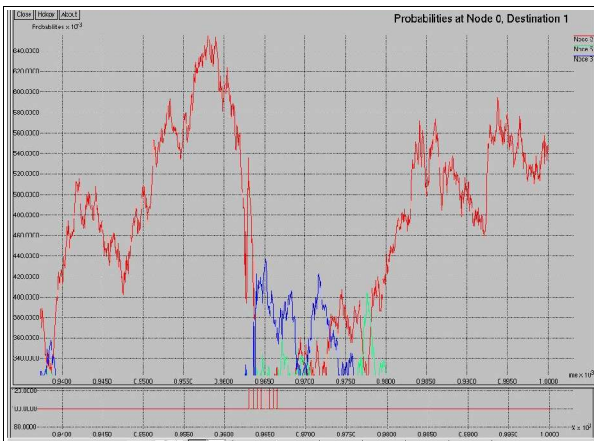


Figure 4: Graph to show improved speed of response with Agent-manipulation on Network Two

Figure 3 shows two overlaid graphs for Network Two. Again the top one shows part of the probability graph (mainly the highest probability); underneath is shown the age function produced by the agent. It can clearly be seen that when there is major disruption in the probabilities - indicating congestion - the age is changed by the agent (which shows as a solid-looking block), speeding up the convergence time.

Figure 4 also consists of two graphs but shows one portion of congestion in more detail. The time before the route is switched in this case is 3.9s, as compared with 4.2s for the ordinary ants, and 3.7s in the case of the ideal-agent. The threshold solution took, anomalously, 4.3s, as it was expected to improve on the ordinary Ant-System.

9.2 Network Three

Results from Network Three show a dramatic improvement in the performance of the Agent-managed Ants over the ordinary system. Due to the larger size of the network, the route-change takes longer in the first place – 9.1s for the simple Ant-Routing system to respond. With the Agent operating however, the changeover took only 6.4s; a significant improvement. Figure 5 shows the graphs of the result.

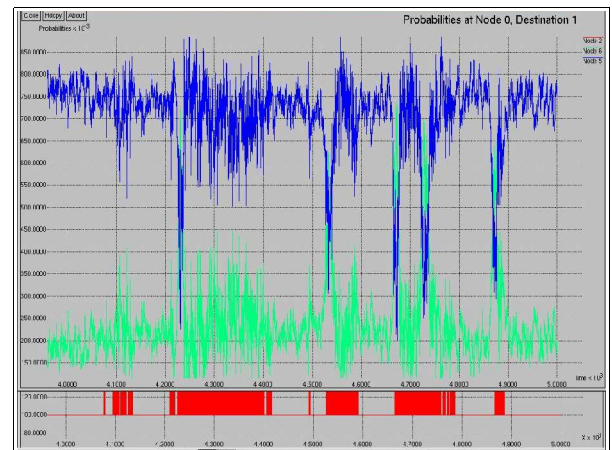


Figure 5: Graph of probabilities and Agent-manipulated Ant-parameters on Network Three

The hard-coded solution took 7.1s to respond fully to the congestion, which suggests that the learning system adds effectiveness to the Agent, whereas the reactive agent only improved on the Ant-System by 0.1s, at 9.0s.

9.4 Overall Results

It has been shown that the Agent is able to respond to changing conditions on the network and act accordingly. It is able to learn the correct circumstances in which to change the response speed of the ants, both increasing and decreasing it.

In comparison, it outperformed both the ordinary Ant-Based Routing System, and the simple reactive Agent, but not the Ideal Agent – which of course would not be realisable (outside these simulations).

It can be seen from the graphs presented that the changing of the Ant-Parameters has a significant affect on the “noise” present in the system, and this could lead to a destabilising effect, causing Route-Flapping, and oscillatory problems across the network.

10 Further Work

The Agent is currently able to manipulate a single parameter strategically; this was for the sake of simplicity, but it could easily be envisaged that more parameters could also be adapted; such as the rate at which pheromone is dropped, and ultimately the rate at which ants are produced. This, as already noted is a double-edged sword, as the ants may start to contribute to the congestion itself.

Secondly, there is potential for a further agent to operate on the actual routing decision – note that the purpose of an Ant-System is to measure the performance of different routes, while the Agent described here changes its speed of response. The Ant-System in affect “recommends” a route and this further agent could apply reasoning to the process – to stop the route “flapping” and to provide coordination which would prevent routing-loops; as alluded to in [Kapetanakis 2004], this would represent a *massive co-ordination step*.

11 Conclusions

The Ant-Based System has been shown to find optimal routing solutions to static conditions, but has been found to show undesirable characteristics when put under the strain of dynamic loading.

A solution has been sought that utilises a software agent to adapt to these changing conditions by manipulating a parameter that controls the speed of response of the Ant-System.

The solution has been implemented, and utilises a Neural Network to perform Q-Learning and choose the optimum action for the state of the network.

This Agent has been compared with an Ordinary Ant-Based Routing System, and two similar Agent-managed Ant Systems, one representing an Ideal Agent, and the other a Simple Reactive Agent. As expected the developed Agent, outperforms all but the Ideal Agent.

References

Breslau L., Estrin D., Fall K., Floyd S., Heidemann J., Helmy A., Huang P., McCanne S., Varadhan K., Xu Y., Yu H., “Advances in Network Simulation”, IEEE Computer, 33(5), pp. 59-67, May, 2000.

Brooks R.A., “A Robust Layered Control System for a Mobile Agent”, IEEE Journal of Robotics and Automation, 2(1), pp:14-21, March 1986.

Dorigo M. & Di Caro G., “AntNet: A Mobile Agents Approach to Adaptive Routing”, Technical Report IRIDIA 97-12, 1997.

Kapetanakis S., Kudenko D., “Improving on the Reinforcement Learning of Coordination in Cooperative Multi-Agent Systems”, Proc. AAMAS-II, AISB'02, pp 89-94, Imperial College, London, April 2002.

Kapetanakis S., Kudenko D., Strens M., “Learning to Coordinate using Commitment Sequences in Cooperative Multi-Agent Systems”, AISB Journal, 1(5), 2004.

Schoonderwoerd, R., Holland, O., Bruten, J., “Ant-Like Agents for Load Balancing in Telecommunications Networks” Proc. 1st Int. Conf. on Autonomous Agents, Marina del Rey, pp 209-216, ACM Press, 1997.

Sutton R.S., Barto A.G., “Reinforcement Learning: An Introduction” MIT Press, 02-6219398-1

Tesauro G., “Programming backgammon using self-teaching neural nets”, Artificial Intelligence, 134 (1-2): 181-199 January 2002.

Tesauro, G., “Pricing in Agent Economies using Neural Networks and Multi-Agent Q-Learning”, Proc. Workshop ABS-3 “Learning About, From and With other Agents”, August 1999.

Vittori K. & Araújo F.R., “Agent-Oriented Routing in Telecommunications Networks”, IEICE Transactions on Communications, E84-B(11):3006-3013, November 2001.

A Motivation-based Approach for Autonomous Generation and Ranking of Goals in Artificial Agents

Luís Macedo^{1,2}

¹Department of Informatics and Systems Engineering,
Engineering Institute, Coimbra Polytechnic Institute
R. Pedro Nunes, Quinta da Nora, 3030-199 Coimbra -
Portugal
lmacedo@isec.pt

Amílcar Cardoso²

²Centre for Informatics and Systems of the University
of Coimbra
Pinhal de Marrocos, 3030 Coimbra - Portugal
{macedo,amilcar}@dei.uc.pt

Abstract

In this paper we describe an architecture of an artificial agent that is able to autonomously generate and rank its own goals or intentions based on its motivations. We present an experiment conducted in a simulated environment with such an agent.

1 Introduction

Considered by many authors as the principal motivational system, emotion is one of the sub-systems that compose personality (Izard, 1991), a characteristic that agents may exhibit (Etzioni & Weld, 1995). Another important sub-system is the drive system (also an important kind of the motivational system). Psychological and neuroscience research over the past decades suggests that emotions play a critical role in decision-making, action and performance, by influencing a variety of cognitive processes (e.g., attention, perception, planning, etc.). Actually, on the one hand, recent research in neuroscience (Damasio, 1994) supports the importance of emotions on reasoning and decision-making. On the other hand, there are a few theories in psychology relating motivations (including drives and emotions) to action (Izard, 1991). For instance, in the specific case of emotions, within the context of the belief-desire theories of action (the dominant class of theories in today's motivation psychology) there have been proposals (Reisenzein, 1996) such as that emotions are action goals, that emotions are or include action tendencies, that emotions are or include goal-desires, and that emotions are mental states that generate goal-desires.

Another important characteristic that agents should also exhibit is autonomy (Etzioni & Weld, 1995). In order to be autonomous, agents should be able to generate their own goals and state preferences between them.

In this paper we describe an artificial agent that is able to autonomously generate and rank its own goals or intentions based on its motivations.

The next section presents an overview of the agent's architecture, giving special attention to the deliberative reasoning/decision-making module in which the generation and ranking of goals are included. Finally, a qualitative experiment is described, discussed and some conclusions are achieved.

2 Agent's Architecture

The architecture that we adopted for an agent (Figure 1) is based on the belief, desire, and intention (BDI) approach (Rao & Georgeff, 1995). Besides, the agent is of motivational kind, exhibiting a module of emotions, drives and other motivations. These play a central role in reasoning and decision-making since they may be thought as action goals (Reisenzein, 1996). The next subsections describe in more detail the main modules of the architecture. The information of the environment is provided to these modules by the sensors, and the effectors execute the actions selected.

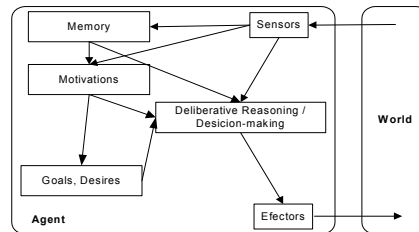


Figure 1. Architecture of an agent.

2.1 Memory

The memory of an agent stores information about the world. This information includes the configuration of the surrounding world such as the position of the entities (objects and other animated agents) that inhabit it, the description of these entities themselves, descriptions of the sequences of actions (plans) executed by those entities and resulting from their interaction, and, in generally, beliefs about the world. This information is stored in several memory components. Thus, there is a metric (grid-based) map (Thrun, 2002) to spatially model the surrounding physical environment of the agent. Descriptions of entities (physical structure and function) and plans are stored both in the episodic memory and in the semantic memory (Aitkenhead & Slack, 1987). We will now describe in more detail each one of these distinct components.

2.1.1. Metric Map

In our approach, a (grid-based) metric map of the world is a three-dimensional grid in which a cell contains the information of the set of entities that may alternatively occupy the cell and the probability of this occupancy. Thus, each cell $\langle x, y \rangle$ of the metric map of an agent i is set to a set of pairs $\phi_{x,y}^i = \{ \langle p_1^i, E_1^i \rangle, \langle p_2^i, E_2^i \rangle, \dots, \langle p_n^i, E_n^i \rangle, \langle p_{n+1}^i, 0 \rangle \}$, where E_j^i is the identifier of the j^{th} entity that may occupy the cell $\langle x, y \rangle$ of the metric map of agent i with probability p_j^i

$\in [0,1]$, and such that $\sum_{j=1}^{n+1} p_j^i = 1$. Note that the pair

$\langle p_{n+1}^i, 0 \rangle$ is included in order to express the probability of the cell being empty. Cells that are completely unknown, i.e., for which there are not yet no assumptions/expectations about their occupancy, are set with an empty set of pairs $\phi_{x,y}^i = \{\}$. Note also that each entity may occupy more than a single cell, i.e., there might be several adjacent cells with the same E_j^i .

2.1.2. Memory for Entities

The set of descriptions of entities perceived from the environment are stored in the *episodic memory of entities*. Each one of these descriptions is of the form $\langle ID, PS, F \rangle$, where ID is a number that uniquely identifies the entity in the environment, PS is the physical structure, and F is the function of the entity. The sensors may provide incomplete information about an entity (for instance, only part of the physical structure may be seen or the function of the entity may be undetermined). In this case the missing information is filled in by making use of the conditional probabilistic Bayes's rule (Shafer & Pearl, 1990), i.e., the missing information is estimated taking into account the available information and descriptions of other entities previously perceived and already stored in the *episodic memory of entities*. This means some of the descriptions of entities stored in memory are uncertain or not completely known (e.g.: element 4 of Figure 2).

The physical structure of an entity may be described analogically or propositionally (Aitkenhead & Slack, 1987). The analogical representation reflects directly the real physical structure while the propositional representation is a higher level description (using propositions) of that real structure.

The analogical description of the physical structure of an entity comprises a three-dimensional matrix and the coordinates of the gravity centre relatively to the entity and to the environment spaces. Notice that the three-dimensional matrix of the entity is a submatrix of the matrix that represents the metric map.

The propositional description of the physical structure of an entity relies on the representation through

semantic features or attributes much like in semantic networks or schemas (Aitkenhead & Slack, 1987). Entities are described by a set of attribute-value pairs that can be graph-based represented.

The function is simply a description of the role or category of the entity in the environment. For instance, a house, a car, a tree, etc. Like the description of the physical structure, this may be probabilistic because of the incompleteness of perception. This means, this is a set $F = \{ \langle \text{function}_i, \text{prob}_i \rangle : i=1, 2, \dots, n \}$, where n is the number of possible functions and $P(\text{"function"} = \text{function}_i) = \text{prob}_i$.


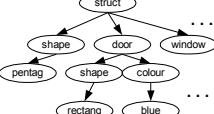

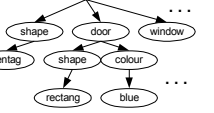

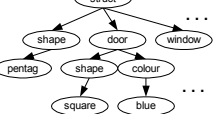

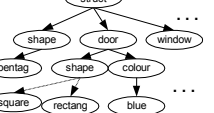
Id	Analogical	Propositional	Function
1			$\langle \text{house}, 1.0 \rangle$
2			$\langle \text{church}, 1.0 \rangle$
3			$\langle \text{house}, 1.0 \rangle$
4			$\langle \text{house}, .066 \rangle$ $\langle \text{church}, .033 \rangle$

Figure 2. Example of the episodic memory of entities. Although the matrix of the analogical description is of three-dimensional kind, for the sake of simplicity, it is represented here as a two-dimensional matrix corresponding to the upper view of the entity.

Concrete entities (i.e., entities represented in the episodic memory) with similar features may be generalized or abstracted into a single one, an abstract entity, which is stored in the semantic memory for entities.

2.1.3. Memory for Plans

Like entities, we may distinguish two main kinds of plans: concrete plans, i.e., cases of plans (Kolodner, 1993), and abstract plans.

We represent plans as a hierarchy of tasks (a variant of HTNs) (e.g., (Erol, Hendler, & Nau, 1994)) (see Figure 3). Formally, a plan is a tuple $AP = \langle T, L \rangle$, where T is the set of tasks and L is the set of links. This structure has the form of a planning tree, i.e., it is a kind of AND/OR tree that expresses all the possible ways to decompose an initial task network. Like in regular HTNs, this hierarchical structure of a plan comprises primitive tasks or actions (non-decomposable tasks) and non-primitive tasks (decom-

posable or compound tasks). Primitive tasks correspond to the leaves of the tree and are directly executable by the agent, while compound tasks denote desired changes that involve several subtasks to accomplish it. Tasks that are the roots of HTN plans are called *goal tasks*. For instance, the leaf node *PTRANS* of Figure 3 is a primitive task, while *visitEntity* is a compound task (and also a goal task).

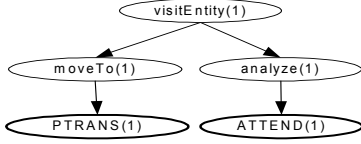


Figure 3. A simple example of plan. Primitive tasks are represented by thick ellipses while non-primitive tasks are represented by thin ellipses.

A task T is both conditional and probabilistic (e.g.: (Blythe, 1999)). This means each task has a set of conditions $C = \{c_1, c_2, \dots, c_m\}$ and for each one of these mutually exclusive and exhaustive conditions, c_i , there is a set of alternative effects $\mathcal{E} = \{ \langle p_1^i, E_1^i \rangle, \langle p_2^i, E_2^i \rangle, \dots, \langle p_n^i, E_n^i \rangle \}$, where E_j^i is the j^{th} effect triggered with probability $p_j^i \in [0,1]$ by condition c_i (i.e.,

$P(E_j^i | c_i) = p_j^i$), and such that $\sum_{j=1}^n p_j^i = 1$. Figure 4 pre-

sents the structure of a task. The probabilities of conditions are represented in that structure although we assume that conditions are independent of tasks. Thus, $P(c_i | T) = P(c_i)$. The main reason for this is to emphasize that the Expected Utility (EU) of a task, in addition to the probability of effects, depends on the probability of conditions too. In addition to conditions and effects, a task has other information components.

Each effect comprises itself a few components of several kinds such as temporal, emotional etc. These components may be of two kinds: non-procedural (factual) and procedural. The non-procedural component refers to the data collected from previous occurrences of the effect (contains the duration of the task, the emotions and respective intensities felt by the agent, the fuel consumed, etc., in previous executions of the task as stored in cases of plans). The procedural component refers to the process through which the temporal, emotional and other kinds of data may be computed (contains descriptions or rules of how to compute the components).

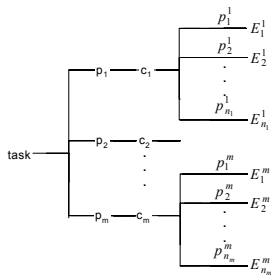


Figure 4. Schematic representation of a task in an abstract plan.

2.2 Motivations

This module receives information from the current state of the environment and outputs the intensities of motivations (emotions, drives and other motivations). In this paper, this module is confined to the motivations that are related with variables that directly influence the main activities that the agent exhibits (exploration and creativity¹): surprise (elicited by unexpectedness), curiosity (elicited by novelty). In addition, we also consider the influence of the drive “hunger” that reflects the need of a power source. Nonetheless, other emotions, drives and other motivations may be included in this module but not considered for the purpose of this paper.

The agent is almost continuously presented with an *input proposition* (Ortony & Partridge, 1987), which corresponds to some sensorial information of an entity (for instance, “a house with squared windows”). In response to this external stimulus, the surprise and curiosity unit outputs the intensity of these motivations, respectively.

In what concerns to surprise, we have developed a computational model (Macedo & Cardoso, 2001a) with the collaboration of the psychologists of the University of Bielefeld, Germany (Meyer, Reisenzein, & Schützwohl, 1997), and also based on the ideas of Ortony and Partridge (Ortony & Partridge, 1987). The idea behind this model is that surprise consists of the appraisal of unexpectedness. Actually, there is experimental evidence supporting that the intensity of felt surprise increases monotonically, and is closely correlated with the degree of unexpectedness (see (Macedo & Cardoso, 2001a) for more details). This means that unexpectedness is the proximate cognitive appraisal cause of the surprise experience. Considering this evidence, we have already proposed (Macedo & Cardoso, 2001a) that the surprise felt by an agent Agt elicited by an object Obj_k is given by the degree of unexpectedness of Obj_k , considering the set of objects present in the memory of the agent Agt , which is given by the improbability of Obj_k (see (Macedo & Cardoso, 2001a) for more details):

$$\begin{aligned} SURPRISE(Agt, Obj_k) &= \\ UNEXPECTEDNESS(Obj_k, Agt(Mem)) &= 1 - P(Obj_k) \end{aligned}$$

We define curiosity (following McDougall (McDougall, 1908), Berlyne (Berlyne, 1950) and Shand (Shand, 1914)) as the desire to know or learn an object that arouses interest by being novel, which means that novel objects stimulate actions intended to acquire knowledge about those objects. Thus, if we accept the above definition, the curiosity induced in an agent Agt by an object Obj_k depends on the novelty or difference of Obj_k relatively to the set of objects present in the memory of Agt :

$$CURIOSITY(Agt, Obj_k) = DIFFERENCE(Obj_k, Agt(Mem))$$

¹ The agents that we have implemented have been used to explore unknown environments (Macedo & Cardoso, 2001b), and to create things (Macedo & Cardoso, 2001c).

The measure of difference relies heavily on error correcting code theory (Hamming, 1950): the function computes the distance between two objects represented by graphs, counting the minimal number of changes (insertions and deletions of nodes and edges) required to transform one graph into another.

The drive hunger is defined as the need of a source of energy. Given the capacity C of the storage of that source in an agent, and L the amount of energy left ($L \leq C$), the hunger elicited in an agent is computed as follows:

$$HUNGER(Agt) = C - L$$

2.3 Goals/Intentions and Desires

Desires are states of the environment the agent would like to happen, i.e., they correspond to those states of the environment the agent prefers. This preference is implicitly represented in a mathematical function that evaluates states of the environment in terms of the positive and negative feelings they elicit in the agent. This function obeys to the Maximum Expected Utility (MEU) principle (Russel & Norvig, 1995). The agent prefers always those states that make it feel more positive feelings (more positive emotions and the satisfaction of drives). Goals or intentions may be understood as something that an agent wants or has to do. These might be automatically generated by the agent or given by other agents.

2.4 Deliberative Reasoning/Decision-making

The reasoning and decision-making module receives information from the internal/external world and outputs an action that has been selected for execution. Roughly speaking, the agent starts by computing the current world state. This is performed taking into account the information provided by the sensors (which may be incomplete) and generating expectations or assumptions for the missing information. Assumptions and expectations for the current agent's position are also generated. The agent has a queue of goal tasks/intentions ranked by their priority (i.e., EU). The first of the ranking is the goal/intention that is under achievement. Once one goal is achieved, it is removed from the queue and the way it was achieved could be learned for future reuse by simply storing its plan in memory as a case. However, external events or objects, for instance, may give rise to new goals/intentions. This is the next step of the reasoning/decision-making process: the generation of new intentions/goals, computation of their EU and insertion of them in the queue of goals/intentions according to their priority (i.e., their EU). Though, if the queue was empty before this step and no new goals are generated in this step, the queue remains empty. In this case there is nothing to reasoning or deciding about and consequently no action is returned. However, the most likely is that the queue is not empty either before or after the step of generating

new goals. If the first goal of the queue is still the same then proceed with its execution and possibly replanning if necessary. However, the addition of new goals may have caused changes in the ranking of the goals in the queue because a new goal may be more EU than some old goals. Thus, the first goal may now be different from the previous first goal. In this case the old first goal is considered suspended. This suspension could happen even though the goal was already under achievement (there was already a plan built for it and this plan was already being executed). Thus, a plan is required for this new first goal in queue, which will be from now on the current goal until its achievement or suspension. That plan could be built or retrieved from memory (if there is one – remember that this current goal may be previously suspended or even previously achieved in the past).

The generation of plans is performed much like in HTN approaches (see (Erol et al., 1994)). We will now describe in more detail the step related with the generation and ranking of agent's goals.

2.4.1. Generation and Ranking of Goals/Intentions

The motivational system plays an important role in the generation and ranking of goals/intentions. Actually, according to psychologists, motivations are the source of goals in several manners: these goals may be included in emotions (e.g., when an agent feels anger about something, a possible triggered goal might be fisting the entity that is on the origin of the anger), or emotions may be themselves the goals (e.g., an agent looks for states of the world that elicit certain positive emotions such as happiness or surprise). Therefore, an agent selects actions or sequences of actions that lead to those states of the world. For instance, an agent establishes the goal of visiting an object that seems beforehand interesting (novel, surprising) because visiting it will probably make it feel happy. The algorithm for the generation and ranking of goals/intentions is as follows (see Figure 5). First, the set of different goal tasks present in the memory of plans are retrieved and, for each kind, a set of new goals (*newGoals*) is generated using the function *adaptGoal()*. This function takes as input a goal task retrieved from a plan in the memory of plans, the memory and the perception of the agent, and generates similar goals resulting from the adaptation of the past goal to situations of the present state of the world. The adaptation strategies used are mainly substitutions (Kolodner, 1993). Thus, for instance, suppose the goal task *visitEntity(e7)* is present in the memory of the agent. Suppose also that the agent has just perceived three entities present in the environment, *e1*, *e2* and *e3*. The entity to which *visitEntity* is applied (*e7*) may be substituted by *e1*, *e2* or *e3*, resulting three new goals: *visitEntity(e1)*, *visitEntity(e2)*, *visitEntity(e3)*. Then, the EU of each goal task is computed. As said above, a task *T* is both conditional and probabilistic (e.g.: (Blythe, 1999)). Thus, the execution of a goal task under a given condition may be seen according to Utility Theory as a lottery (Russel & Norvig, 1995):

$$Lottery(T) = \left[p^1 \times p_1^1, E_1^1; p^1 \times p_2^1, E_2^1; \dots; p^m \times p_{n_m}^m, E_{n_m}^m \right]$$

, where p^i is the probability of the condition c_i , p_j^i is the probability of the j^{th} effect, E_j^i , of condition c_i .

The EU of T may be then computed as follows:

$$EU(T) = \sum_{k,j} p^k \times p_j^k \times EU(E_j^k)$$

The computation of $EU(E_j^k)$ is performed predicting the emotions that could be elicited by achieving/executing the goal task. This means, the emotions, drives and other motivations felt by the agent when the effect takes place are predicted or estimated based on the procedural or non-procedural components of the effect.

The following function is used to compute $EU(E_j^k)$:

$$\begin{aligned} EU(E_j^k) &= \\ &= \frac{\alpha_1 \times U_{surprise}(E_j^k) + \alpha_2 \times U_{curiosity}(E_j^k) + \alpha_2 \times U_{hunger}(E_j^k)}{\sum_i \alpha_i} = \\ &= \frac{\alpha_1 \times SURPRISE(E_j^k) + \alpha_2 \times CURIOSITY(E_j^k) + \alpha_2 \times HUNGER(E_j^k)}{\sum_i \alpha_i} \end{aligned}$$

, where, $\alpha_2 = -1$ and α_i ($i \neq 2$) may be defined as follows:

$$\alpha_i = \begin{cases} 1 & \Leftarrow C - HUNGER(Agt) - D > 0 \\ 0 & \Leftarrow \text{otherwise} \end{cases}$$

, where D is the amount of energy necessary to go from the end location of goal task T to the closer place where energy could be recharged, and C is the maximum amount of energy that could be stored by the agent.

In the case of exploratory and creativity behaviour, the surprise and curiosity of an effect of a task are elicited by the objects that the agent perceives.

Algorithm generateRankGoals(*newRankedGoals*)

Output: *newRankedGoals* – the set of ranked goals

```

newGoals ← ∅
setPastGoals ← {x: x is a goal task belonging to some plan in memory}
for each goal in setPastGoals do
    adaptationGoal ← adaptGoal(goal, agtMemy, agtPercepts)
    newGoals ← newGoals ∪ adaptationGoals
end for each
for each goal in newGoals do
     $EU(T) = \sum_{k,j} p^k \times p_j^k \times EU(E_j^k)$ 
end for each
insert(goal, newRankedGoals)
return newRankedGoals
end

```

Figure 5. Algorithm for the generation and ranking of goals.

3 Qualitative Experiment

We have conducted an experiment in order to evaluate the reasoning/decision-making process of an agent with the architecture described above. Special attention was given to the algorithm of the autonomous generation and ranking of goals based on motivations. To do so, we ran an agent in a simulated environment populated with several buildings (their functions were for instance, house, church, hotel, etc.; for the sake of simplicity, their descriptions were related with the shapes of their structure: rectangular, squared, etc.). Figure 6 presents the simulated environment and the path taken by the agent to explore it. The agent started at location 0, with an empty memory of entities, but with a single case of a past plan for visiting entities. At this location its visual field included objects $E1$ and $E2$, located respectively at locations 1 and 2. Then the agent generated goals for visiting them by adapting the goal *visitEntity* of the previous plan stored in memory. The resulting goals are: *visitEntity(E1)* and *visitEntity(E2)*. $E1$ and $E2$ are entirely new for the agent (remember that the agent started with an empty memory of entities). Therefore, the surprise and curiosity that they may elicit when visited is maximum (i.e., 1.0). However, $E1$ is closer, so the hunger that may be felt when the agent is at location 1 is lower than in location 2. Hence, the agent ranks the goals as follows: *visitEntity(E1)* followed by *visitEntity(E2)*. A plan is generated for the first goal. After its execution, the agent is at location 1 with a complete description of $E1$ stored in memory as a case (case 1 of the episodic memory of Figure 2) and an incomplete description of $E2$ (because it has not been visited yet and therefore it is not completely known – at least the function is still undetermined). In addition, the goal *visitEntity(E1)* is deleted from the queue of goals. At location 1, the agent perceives $E2$ and $E3$ ($E1$ is also perceived, but it has just been visited). The agent generates the goal *visitEntity(E3)* for visiting $E3$. Notice that *visitEntity(E2)* is still in the queue of goals. $E3$ is similar to the previously visited $E1$ and therefore it predicts feeling a low intensity of surprise and curiosity when visiting it. Besides, hunger is expected to be higher in location 3 than in 2. So, the goals are ranked as follows: *visitEntity(E2)* followed by *visitEntity(E3)*. Once again, a plan is generated for *visitEntity(E2)* and then executed. The result is the completion of the description of $E2$ (case 2 of the episodic memory of Figure 2). At location 2, the agent perceives $E4$, in addition to $E3$. $E4$ is similar to both $E1$ and $E2$. However, its EU is lower than that of $E3$ mainly because the agent expects a higher hunger in location 4 than in 3. Thus, $E3$ is visited. At this time, the agent has the episodic memory of Figure 2. An interesting behaviour is observed later when the agent has to select between visiting $E11$ and $E12$, which are exactly equal to $E1$ and $E2$, respectively, and at similar distances. Therefore, it might be expected that the agent would visit $E11$. However, this time the agent ranks the goals as follows: *visitEntity(E12)* and *visitEntity(E11)*. This is because the agent has now more cases

describing entities similar to *E11* than to *E12*. Therefore, *E12* is expected to elicit more surprise than *E11*, and hence the EU of visiting *E12* is higher than that of visiting *E11*.

In order to take conclusions about the quality of this behaviour, we asked a few humans to describe the path they would follow in such environment. We verified that there is much similarity with the path followed by the agent.

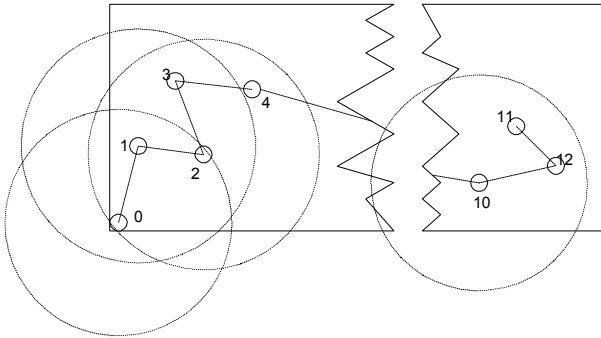


Figure 6. Experiment in a simulated environment. Dashed circles represent the visual field of the agent in different locations.

4 Conclusions

We have presented a motivation-based approach for the autonomous generation and ranking of goals. This approach is in the core of the reasoning process of agents. The experiment conducted allows us to conclude that the behaviour of an agent whose reasoning process includes this approach is similar to that of humans in the simulated environment considered.

Aknowledgements

The PhD of Luís Macedo is financially supported by PRODEP III.

References

- Aitkenhead, A., & Slack, J. (1987). *Issues in cognitive modelling*. London: Lawrence Erlbaum Associates.
- Berlyne, D. (1950). Novelty and curiosity as determinants of exploratory behavior. *British Journal of Psychology*, 41, 68-80.
- Blythe, J. (1999). Decision-Theoretic Planning. *AI Magazine, Summer 1999*.
- Damásio, A. (1994). *Descartes'error, Emotion Reason and the Human Brain*. New York: Grosset/Putnam Books.
- Erol, K., Hendler, J., & Nau, D. (1994). UMCP: A sound and complete procedure for hierarchical task-network planning. *Proceedings of the International Conference on AI Planning Systems* (pp. 249-254).
- Etzioni, O., & Weld, D. (1995). Intelligent agents on the internet: fact, fiction, and forecast. *IEEE Expert*, 10(44-49).
- Hamming, R. (1950). Error Detecting and Error Correcting Codes. *The Bell System Technical Journal*, 26(2), 147-160.
- Izard, C. (1991). *The Psychology of Emotions*. NY: Plenum Press.
- Kolodner, J. (1993). *Case-Based Reasoning*. San Mateo, CA: Morgan-Kaufmann.
- Macedo, L., & Cardoso, A. (2001a). Modelling Forms of Surprise in an Artificial Agent. In J. Moore & K. Stenning (Eds.), *Proceedings of the 23rd Annual Conference of the Cognitive Science Society* (pp. 588-593). Mahwah, NJ: Erlbaum.
- Macedo, L., & Cardoso, A. (2001b). SC-EUNE - Surprise/Curiosity-based Exploration of Uncertain and Unknown Environments., *Proceedings of the AISB'01 Symposium on Emotion, Cognition and Affective Computing* (pp. 73-81). York, UK: University of York.
- Macedo, L., & Cardoso, A. (2001c). Using Surprise to Create Products that get the Attention of other Agents. In L. Canãmero (Ed.), *Emotional and Intelligent II: The Tangled Knot of Social Cognition - Papers from the 2001 AAAI Fall Symposium (Technical Report FS-01-02)* (pp. 79-84). Menlo Park, CA: The AAAI Press.
- McDougall, W. (1908). *An introduction to social psychology*. London: Methuen.
- Meyer, W., Reisenzein, R., & Schützwohl, A. (1997). Towards a process analysis of emotions: The case of surprise. *Motivation and Emotion*, 21, 251-274.
- Ortony, A., & Partridge, D. (1987). Surprisingness and Expectation Failure: What's the Difference?, *Proceedings of the 10th International Joint Conference on Artificial Intelligence* (pp. 106-108). Los Altos, CA: Morgan Kaufmann.
- Rao, A., & Georgeff, M. (1995). BDI Agents: From Theory to Practice, *Proceedings of the First International Conference on Multiagent Systems*.
- Reisenzein, R. (1996). Emotional Action Generation. In W. Battmann & S. Dutke (Eds.), *Processes of the molar regulation of behavior*. Lengerich: Pabst Science Publishers.
- Russel, S., & Norvig, P. (1995). *Artificial Intelligence - A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall.
- Shafer, G., & Pearl, J. (Eds.). (1990). *Readings in Uncertain Reasoning*. Palo Alto, CA: Morgan Kaufmann.
- Shand, A. (1914). *The foundations of character*. London: Macmillan.
- Thrun, S. (2002). Robotic mapping: A survey. In G. Lakemeyer & B. Nebel (Eds.), *Exploring Artificial Intelligence in the New Millenium*. San Mateo, CA: Morgan Kaufmann.

Self-adaptive network of cooperative neuro-agents

J.P. Mano, P.Glize

IRIT, Université Paul Sabatier, 118 route de Narbonne 31062 Toulouse cedex, France

Abstract: According to the most recent research on adaptive multi-agent systems, it is possible to assess that those systems are able of building by themselves a representation of their surrounding world.

Using cooperation as a local criterion of self-organisation it become possible to make a multi-agent system learn how to interpret the different signals it receives in order to adapt its functioning to its environment. After an adaptation period, a complex system is able to emerge by the way of a self-organised learning network.

Key words: Network learning, Knowledge Acquisition, Experimental study, Multi-agent systems, Cognitive Modelling, Functional Emergence.

1 Introduction

If the human being can precisely describe the behaviour of an entity at its level of functioning, it stays quite puzzled understanding in detail how complex system function emerges from its parts activity. As a real gap exists between cellular biology and psychology, a complex system cannot be realised from its level of functioning, but our working hypothesis tells minimal entities of such a system can be built and given with behavioural potentialities in order to organise themselves to satisfy the whole system activity.

In this study, we want to show that a coherent and useful system can emerge without any global model for its functioning, solely from its parts activity and organisation. Our thought process supposes four distinct steps:

- Firstly the neuro-agent unit has to be designed, in fact only agents must be designed,
- Secondly an initial system is constituted of a minimal number of neuro-agents randomly linked or unlinked,
- Thirdly the global system has to be plunged in the environment in which it will have to adapt to.
- Fourthly, through an *a posteriori* study, the internal organization of the global system has to be analysed in order to understand its functioning.

This paper presents after some definitions and a brief methodological presentation, our preliminary results using a multi-agent system of neuro-agents. Then we propose a theoretical discussion on the organisation of such multi-agent systems.

2 Definitions and background

“Essentially, ontogeny is the process that maps the genotype to the phenotype. This mapping is quite simple in most current evolutionary algorithms, though

some more complex (ontogenetic) ones have been proposed.” (Sipper, 1999). It is not yet possible integrating the complexity of the dynamic that occurs during the self-structuring ontogeny of a biological system (Maturana, 1994). Neural structures may be considered as the combined result of self-organising cellular activities and of the following of many strong planned processes. Such a system is the result of the permanent reorganisation of its parts upon the pressure of its environment.

2.1 From AMAS...

In our team, this perception of complex system is transposed in the design of multi-agent systems. Therefore, the conditions required for a complex system to adequately adapt itself, are a coherent local activity of its parts, and a sufficient feedback between system activity in the environment and related perceptions of the system.

The first aim of the Adaptive Multi-Agent System theory (Camps, 1994) is to realize MAS having the “classical” characteristics to build a society of situated agents (Wooldridge, 2002). But, from our point of view, a MAS is also plunged into an environment and must reach a behavioural adequacy or a functional one.

We name this property “functional adequacy” and we proved the following theorem (Gleizes, 1999): “For any functionally adequate system, there is at least a cooperative internal medium system which fulfils an equivalent function in the same environment”. Therefore, we focused on the design of cooperation internal medium systems in which agents are in cooperative interactions.

The specificity of the theory lay in that the global function of the system is not coded within the agents, but emerges from their collective behaviour. Each agent possesses the capacity to locally rearrange its interactions with others depending on its individual task. Changing the interactions between agents can

indeed lead to a global level change that induces the modification of the global function. Cooperation-driven self-organisation implying local treatment of non cooperative situations is a mean to optimise system's functioning when a difficulty is encountered.

Our theory of cooperative self-organisation has already been validated in several fields¹: the "tileworld game" (Piquemal, 1995), cooperative information systems in e-commerce (Gleizes, 1999), behavioural simulation about natural and artificial collective intelligence (Topin, 1999), traffic management of a telephonic network (Dotto, 1999), real-time system for flood forecast (Gleizes, 2003). In all these applications, systems have coarse-grained or middle-grained agents. The goal of current project is twofold: first, to give results when having fine-grained agents and second, to develop tools in order to observe and analyse a self-organizing complex network of agents.

2.2 ...to cooperative neuro agents.

Observing previous works in the domain of artificial learning, we can compare our agents with the neurons of ontogenic artificial networks (Fiesler, 1994). For this reason the concept of cooperative neuro-agent (CNA) has been defined and represents an agent having the usual transfer function of a neuron and having moreover a cooperative behaviour according to the laws of the AMAS theory.

At a functioning level a CNA integrate the information carried by incoming links, then this integrated value is transformed using a transfer function in a value of same dimension as inputs. The transfer function is in fact a composed with a three adjustable segments curve.

At a social level, each CNA develops relation with other CNAs of the network, or with the interface of the system. Those relations are realised through links or synapses that connect two CNAs on a activating or inhibiting mode. The Links are characterized by a weight and a confidence. This confidence represents a kind of inertia of the link whereas the weight is adjusted to transmit the correct amount of activation or inhibition between two CNAs.

At the last level, or vegetative functioning, the CNA can determine if it has to proliferate or not, observing the information it receive thanks to its local perception of the network.

In our network, the mother CNA provides all the required instances, which are an exact copy. Nevertheless, the basic transfer function of the mother cell is slightly adjusted in each individual CNA in order to find the better cooperative behaviour in

accordance with its neighbourhood. Some cooperative behaviours are described in the following table.

NCS detection	CNA behaviour
Conflict 1. CPU over used	Favour activity of best self-confident CNAs
Conflict 2. Being connected to survive and a limited amount of inputs and outputs of the system.	Observing all requests of new creation of connection and allow only the best of them.
Concurrency 1. Duplication or Redundancy are detected when a CNA has two of its inputs synchronised	The 2 upstream CNAs have to become one.
Misunderstanding and Ambiguity 1. Uncorrelated inputs or lack of contingency or too weak weights	Each CNA adjust its inputs' weights to have a coherent functioning Disable too weak links
Uselessness 1. Constitutive inactivity	Search new inputs and increase transfer function sensitivity
Uselessness 2. Constitutive activity	Decrease some weights or search of new inputs or increase transfer function sensitivity
Uselessness 3. Stereotypic functioning means a regular pattern of activation	A CNA should have more than one input; this condition prevents the appearance of cyclic structures.
Uselessness 4. Isolated agent has lost its upstream and downstream	This CNA has to disappear.

3 Experimental results

The study of the network is an *a posteriori* process that needs specific tools to understand the reasons why the system behaves as it behave. As a matter of fact, it is the succession of snapshots of the network organisation which will help to analyze the mechanisms of emergence of system's functions.

3.1 Entropy measurement.

As an online interpretation of global connectivity and network stability, we have used an index of entropy. This index is computed from global mean connectivity, which represents not only the number of connection but also weight and stability of those connections.

¹ Home page of Cooperative Multi-Agent Systems: <http://www.irit.fr/SMAC>

$$I = \sum_{i=1}^N \frac{a_i + b_i}{C_i^{CNA} \left(\sum_{j=1}^a C_j^{Link} \times \omega_j + \sum_{k=1}^b C_k^{Link} \times \omega_k \right)}$$

I so define the entropy of a network containing N CNAs, each CNA presenting a inputs and b outputs. C_i^{CNA} and C_k^{Link} define respectively a CNA confidence and a link confidence.

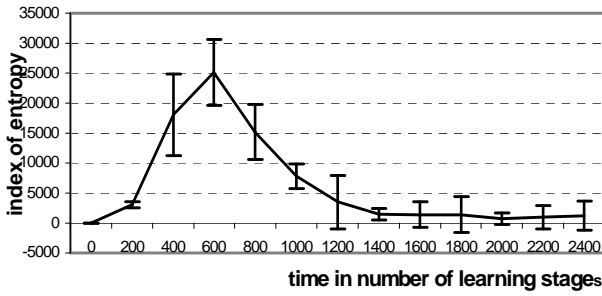


fig 1 : Evolution of the entropy index

The graph (fig. 1) presents the average entropy in a network initially composed by 3 inputs and 3 outputs, each output having to learn an AND logical function upon two randomised inputs. Index of entropy is the result mean weight of connections multiplied by mean number of connections and divided by the mean number of CNAs. The simulation was repeated 60 times; mean and standard deviation are reported. A stabilisation of the global entropy of the system is noticeable after 1400 cycles.

3.2 CNA proliferation and stabilization

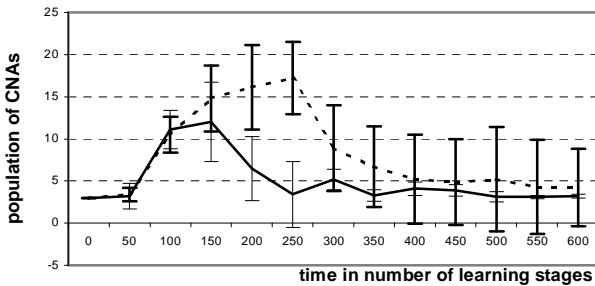


fig 2 : Evolution of the number of CNAs
dotted line represents XOR function learning
single line represents AND function learning

The figure 2 explores the number of CNAs in two series of learning stages. The MAS has to learn logical AND and XOR function; each function was test 20

times. The XOR learning appears effectively harder to do in two dimensions. First, the temporal dimension shows a late in XOR learning about 300 cycles. The second dimension of the issue deals with accuracy of learning. Seeing standard deviation, current CNAs do not perform well each time in XOR learning whereas AND function is sharply and regularly learnt.

3.3 Learning of functional adequacy

The figure 3 shows how the functional adequacy of an output of the MAS is estimated. The single line corresponds to the expected response of the system, and in this case, if observed response is different from expected one, a feed back is applied. The dark rectangle symbolise the period when inputs are active. Curves of fig 3 come from an AND function learning of a five CNAs network and after about a hundred learning cycles.

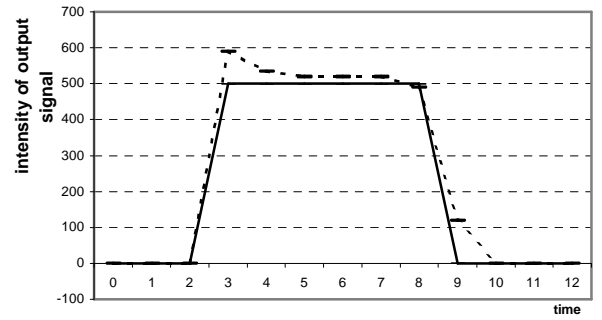


fig 3 : functional adequacy of an output of the MAS

dotted line represents observed output
single line represents expected output

4 Understanding emergence from cooperation

4.1 Building the initial system

In each sub part of the system, the assembling of CNAs is a self-organising process. Self-organisation is characterized here by four conditions:

- When the system is initialized, the only created CNAs without any connections, are situated at the interfaces of the MAS (inputs and outputs),
- Behaviour of CNAs only depends upon their local perception of the system,
- No imposed pattern which leads the organisation of the system,
- No explicit feed back established from the system activity, but only a perception of environment modifications.

During a learning period, the system adjusts its function by reorganising its parts. So the learning of the system is the global result of population growth, of neuro-agents adaptation, in details, weight adjustments, transfer function regulation, research and disappearance of connections.

4.2 Sampling information from the environment

The system has to develop perceptions upon its environment. Commonly, the term of perception refers to the whole cognitive processes occurring between an environmental event and its induction in the mind. Upstream perception is carried by interfaces of the system and by interfaces is meant the sum of inputs agents. Finally the inputs of the system are designed as specialized neuro-agents sensitive to an exclusive dimension of the informative space. For each sensitive channel of the system, the number of “receptive-agent” is determined so that an agent is associated with a single perceptive element. Information is by this way sampled and turned into the agent’s language to be propagated inside the system.

The informative space is though sampled in a multi-dimensional space where a dimension is brought by an input CNA. As an example see the topology of the different sensitive organs of mammals, according to the neural structures of those organs, we have built the sensitive interfaces of our system. So as an eye is sensitive to the topography of the surrounding physical environment and is also sensitive to different colours and light intensities, each photoreceptor reacts selectively to a wavelength (or a range wavelength) and its activity is related with the intensity of the signal.

4.3 Identifying minimal acts

To define the output of the system, we have to design some special CNAs that are actuators of the system. Their role is to transpose the information handled by the network in single acts. These acts are events realisable by the network and susceptible of being perceived in the environment. The number and the nature of these minimal acts are established according to the expected global function. For an external observer it is the coordination of minimal acts driven by the multi-agent system that is understood as system’s behaviour.

4.4 Cooperation-driven Self-organisation

Self-organisation is a mean of leading a non organised system to an organised one, by the way of emergence (Holland, 1997). A primordial condition of emergence is the absence of any pattern that predefines the global

organisation of the system. That’s why neuro agents only have a local view upon their own environment. By the way of cooperation, each neuro agent can share a part of the information it perceives. Being locally “cooperative” does not mean that CNA is always helping the others or that it is altruistic but only that it is able to recognize and treat cooperation failures.

5 Regulation of CNA function

In order to keep in a cooperative way most of the time, each neuro agent will adapt itself to deal with whatever disturbing event. From an egocentric point of view, it adjusts the weights of its inputs, and its transfer function. This regulation seems hebbian reinforcement learning (Floreano, 2001), but the originality presented here consists in the criteria taken into account to quantify weights and other parameters adjustments. This kind of unsupervised and self-organised learning is often used (Linsker, 1986) and different mechanisms of neuron adjustments are proposed.

Reinforcement learning provides a means to design systems without needing to specify how the objectives should be reached. This implicit learning characteristic sets reinforcement learning apart from conventional non-linear systems that require explicit process knowledge (Moore, 1996).

In our study, the CNA main objective is to satisfy, to be useful to the others by having a coherent activity and furnishing to the other relevant information. In this aim, learning consists in reinforcing weights according to correlated activities of inputs. These conditions of contiguity encompass two kinds of events: those having spatial proximity and those presenting temporal correlation. Here we have considered that in this system absolutely self-organised, CNAs are only able to compare the temporal correlation of their incoming pattern of activation’s.

Moreover CNA can modify its function to keep the information it builds intelligible by others neuro agents. Each CNA can adjust its transfer function according to its local perception of the network and of environment. Explicitly that means a CNA modify its functioning to fulfil others CNAs it works with. So at a given instant the behaviour of a CNA is the result of its code expression under the regulation of its neighbourhood.

Considering the population of CNAs we cannot precisely describe it neither as a homogenous population nor as a composition of different types of agent. This on-line adaptation process should give the system a robustness advantage in comparison with neural network using genetic algorithm to adapt the neurons behaviour (Moriarty, 1997).

6 Conclusion

The problem we study here is complex by the way that the system adaptation discontinuous. In fact the MAS is an open system whose parts can appear, disappear, and reorganize their connections autonomously.

An analysis of the system organisation implies two levels of observations. A static point of view upon the final system will permit a comparison with biological even known systems. Otherwise, a dynamical observation should explore the emergence of a function in a complex system.

The training phase allows modifying the strengths between nodes in order to reduce the error between the actual and desired output's signals. In order to overcome their limitations (sub-optimal solutions, off-line training, convergence speed...), lot of works try to add plasticity in the basic neural network architecture. In the same vein, the self-adaptive network of CNA presented here, is able to define criteria for adapting the genotypic transfer function, the node strengths, the connectivity between nodes, the number of layers, and even neuron deaths. In short, the AMAS theory allows creating synthetic cellular networks having the basic epigenetic, phylogenetic and ontogenetic properties.

The local criterion of favouring cooperation between neuro-agent is a guarantee of learning's optimisation.

Acknowledgments

I thank PG and MPG for their continuous support. I am grateful to Daniel Kudenko for inviting me to the workshop on "Adaptive Agents and Multi-Agents Systems"

References

- Camps V., Gleizes M.P., & Glize P., Une théorie des phénomènes globaux fondée sur des interactions locales, 1998, Actes des Sixième journées francophones IAD&SMA Pont-à-Mousson - Editions Hermès
- Fiesler E., Comparative Bibliography of Ontogenic Neural Networks, 1994, Proceeding of the International Conference on Artificial Neural Networks – ICANN, 1994
- Floreano D., Urzelai J., Evolution of plastic control networks, 2001 Kluwer academic publishers.
- Forrest S., Emergent computation : Self-organising, collective, and cooperative phenomena in natural and artificial computing network, 1990, Physica D42 Vol1 N°11 - North-Holland.
- Gleizes M.P., Camps V., Glize P., A theory of emergent computation based on cooperative self-organisation for adaptive artificial systems, 1999, Fourth european congress on systemic, Valence.
- Gleizes M.P., Léger A., Athanassiou E., Glize P., Abrose : Self-Organisation and Learning in Multi-Agent Based Brokerage Services, 1999, 6th International Conference on Intelligence and Services in Networks, Lecture Notes in Computer Science 1597, Springer.
- Gleizes M.P., Régis C., Georgé J.P., Glize P.- Real-time simulation for Flood Forecast, 2003, Third international conference on Autonomous agents and Multi-Agent Systems.
- Goldstein, J., Emergence as a Construct: History and issues, 1999, Emergence Volume 1, Issue 1.
- Holland J.H., Emergence: from Order to Chaos, 1997, Addison-Wesley
- Kaelbling L.P., Littman M.L. & Moore A.W., Reinforcement Learning: A Survey, 1996, Journal of Artificial Intelligence Research.
- Maturana H.R., & Varela F.J., L'arbre de la connaissance, 1994, Addison Wesley.
- Moriarty D.E., & Miikkulainen, R., Forming neural networks through efficient and adaptive co-evolution, 1997, Evolutionary Computation.
- Piquemal-Baluard C., Camps V., Gleizes M.P., Glize P. Cooperative agents to improve adaptivity of multi-agent systems, 1995, Intelligent Agent Workshop of the British Computer Society in Specialist Interest Group on Expert Systems & Representation and Reasoning, Oxford.
- Sipper M., Notes on the Origin of Evolutionary Computation, 1999, John Wiley & Sons, Inc., Complexity Vol. 4, No. 5.
- Topin X., Fourcassie V., Gleizes M.P., Théraulaz G., Régis C. Glize P., Theories and experiments on emergent behaviour : From natural to artificial systems and back, 1999, Proceedings on European Conference on Cognitive Science, Siena.
- Wooldridge M., An introduction to multi-agent systems, 2002, John Wiley & Sons.

A Multi-Agent framework for web information retrieval

Pedro A. C. Sousa

Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa
Caparica, Portugal
pas@fct.unl.pt

João P. Pimentão

Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa
Caparica, Portugal
pim@fct.unl.pt

Bruno R. D. Santos

UNINOVA
Campus da FCT/UNL
Caparica, Portugal
brd@uninova.pt

Adolfo Steiger Garção

UNINOVA
Campus da FCT/UNL
Caparica, Portugal
asg@uninova.pt

Abstract

This paper presents a Multi-Agent based web content categorization system. The system was prototyped using a Framework for Internet data collection. The Framework and its application to E-commerce are presented. The advantages derived from agent's technology usage are presented.

1 Introduction

The proliferation of unwanted information is becoming unbearable, imposing the need of new tools to overcome the situation. Either through push mechanisms, such as Email spamming, or by pull mechanisms as presented in web sites, it is necessary to ensure that the individuals can access the relevant information, avoiding unwanted information; saving valuable time otherwise spent on processing more information than that is needed and to decide which information to retain or discard.

In consequence, tools to enhance the results of the current electronic procedures of acquiring data, substituting them by more efficient processes, are required. The "Framework for Internet data collection based on intelligent agents", hereon designated by "Framework", developed in UNINOVA, offers a user-friendly interface customized through a user's-personalized catalogue, which is automatically updated with information gathered from available web sites. The catalogue stores, in a pre-selected ontology, the data collected from different web sites, avoiding manual useless visits to several sites. The process of presenting the catalogue's-collected data is ergonomic and automates the user's most common tasks.

The autonomous data collection and semi-automatic catalogue updating is executed by a FIPA compliant Multi-Agent System (MAS), relieving the end-user

from all the "hard work" (i.e. interfacing the web and the private databases). The agents increase their performances, taking advantage of text learning methods, dedicated to Internet information retrieval. The use of an Agent-based System was agreed due to its inherent scalability and ease to delegate work among agents. The Framework being presented was intensively tested in the project DEEPSIA. All the examples presented in this paper were produced under DEEPSIA's context; i.e., the creation of personalised catalogues of products sold on the Internet.

1.1 The DEEPSIA project

The project DEEPSIA "Dynamic on-linE IntErnet Purchasing System, based on Intelligent Agents", IST project Nr. 1999-20 283, funded by the European Union has the generic aim of supporting Small and Medium Enterprises (SME) usual day-to-day purchasing requirements, via the Internet, based on a purchaser-centred solution and tailored to meet individual needs.

Usually, Business-to-Business (B2B) e-commerce models focus on SMEs as suppliers, often within virtual shops or marketplaces, nevertheless, all SMEs are purchasers of goods and services. Depending on the size of the SME, this function may be performed by one person the owner/manager, or the purchasing manager – or may be open to a number of staff.(Sousa, Pimentão, Pires and Garção, 2002)

The procurement process is no exception in trying to find relevant information on an information-overloaded web, magnified by the myriad of new commercial sites that are made available everyday.

DEEPSIA's purpose is to provide a purchaser centred solution, available on the user's desktop and tailored to individual requirements, rather than a supplier centred marketplace.

The reduction of time in collecting data will be achieved by presenting the user with a catalogue of products organised under an ontology representing a set of relevant entries, thus avoiding having to browse through a wide range of web pages and links that are not related to his/her needs.

2 The Deepsia's adopted framework

DEEPSIA adopted a framework composed of three subsystems, responsible for performing specific tasks, and interacting with each other through data and information flows.

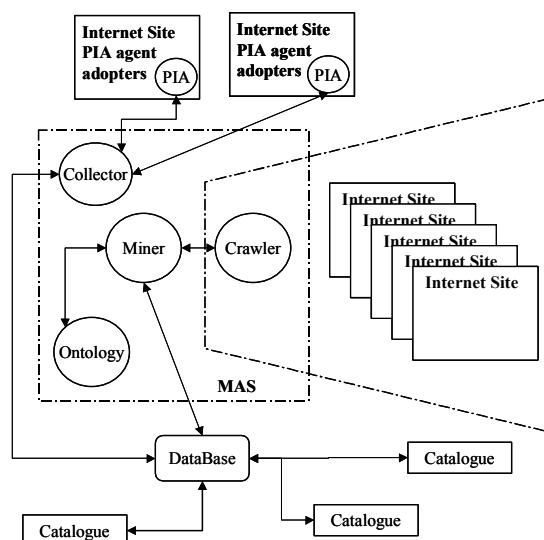


Figure 1: Conceptual description of the most relevant components (represented by rectangles) and software Agents (represented by circles) that compose the Framework.

Depicted in Figure 1, the Framework's subsystems are: a Dynamic Catalogue (DC), responsible for data storage and presentation; a Multi-Agent System (MAS), responsible for autonomous data collection, and semi-automatic catalogue process update; and an optional Portal and Agents Interface System (PAIS),

responsible for interfacing directly with the web data suppliers.

2.1 Dynamic Catalogue

The Dynamic Catalogue is the user interface, which is responsible for presenting the information collected by the MAS based on the user's preferences. The dynamic on-line catalogue communicates with the MAS in order to provide the end-users with all the required information. The catalogue holds not only the information about the data selected by the agents on the web, but also the information about the sites contacted, the ontology in use (configurable by the user) and all user requirements regarding the data gathered and stored. The electronic catalogue's data is stored in a database (DB) and it is made available through a web-browsing interface. For concept storage, ontology and conceptual graph approaches were selected in order to create a user-friendly interface (Tan, 1993).

For the customisation of the architecture to the project, the DEEPSIA's consortium selected the Universal Standard Products and Services Classification (UNSPSC), from ECCMA, as the base taxonomy for the ontology system. ECCMA - Electronic Commerce Code Management Association, presents itself as a not-for-profit, unbiased, membership organisation that oversees the management and development of the UNSPSC Code.(ECCMA, 2002)

The UNSPSC code file being used contains more than 12,000 commodities, spanning fifty-six industry segments. The UNSPSC code has been implemented worldwide within major procurement and financial software systems. It is based on B2B commerce and any size of company throughout the world can benefit from its use.

The dynamic catalogue is filled with information provided using two search mechanisms: The Direct Search and the Autonomous Search.

2.2 Direct Search

The PAIS implement the direct search system, using Portal Agent Interfaces (PIA), adopted by the commercial web sites and a Collector Agent, which is the multi-agents' systems interface to manage the PIAs.

The PIA is responsible for creating an interface between one specific web supplier's portal (Internet

Site PIA agent adopters) and the MAS. The PIA is a facilitator agent installed in the web supplier's site. With this agent, the web supplier creates a dedicated interface to interact with the MAS, thus enabling direct querying to their sites' private databases. It is not mandatory for this agent to be adopted by the web suppliers, since the MAS is able to collect information based on the autonomous search strategy. Nevertheless, its inclusion provides the web suppliers with a much more integrated and flexible interface and enables the possibility of direct querying to web suppliers' DB, i.e. the MAS is able to query the web supplier about specific information.

2.3 Autonomous Search

Focusing in the MAS it would be possible to identify, several types of agents that accomplish subsidiary tasks contributing to the ultimate goals of the system. The tasks performed by the different agents, include between others: message processing, name resolution service, web page discovery, etc. These agents are not described in this paper since their role is quite common and it is not relevant in the context of information retrieval.

The agents responsible for performing the autonomous search can be identified in the Figure 1, following the path that goes through the Web Crawler agent (WCA), the Miner agent (MA), and the database (DB), with the connection with the Ontology agent. The WCA has the responsibility of crawling and identifying pages with the user's interest themes. The MA has the responsibility to identify the relevant concepts included in the pages selected by the WCA, with the support of the Ontology Agent (that stores all the concepts that the system recognises), and send the concepts, together with the ontology classification, to the user's catalogue.

The Ontology agent is responsible for ontology management and for classifying the concepts found. The Ontology agent answers to queries from the miner for product classification purposes.

To keep the Agent's simplicity, all the learning processes (feature selection, classifiers creation, DSS-Decision Support System definition) are performed by the special Tutor Agent, (omitted from figure 1 to improve readability). Their achieved results are mapped to XML and then communicated, using FIPA compliant agent messages, to the respective agents (the Crawler and Miner) (FIPA00061, 2000). The basic behaviour of all agents is based on a bootstrap

knowledge, complemented in time, by the Tutor Agent whenever new conclusions are achieved.

Applying the Autonomous Search Mechanism to product promotions' identification scenario (one objective of DEEPSIA), it will result in the following tasks division. A WCA is the responsible discovering and deciding if a page is a product promotion page or not. The MA is responsible for the product identification and for determining the promotion's associated conditions (price, quality, etc).

2.4 The Web Crawler Agent (WCA)

The WCA automatically fetches web pages looking for pre-selected themes defined by the user. The search starts by fetching a seed page, and then all the pages referenced by the seed page, in a recursive approach. Therefore, each time the end-user finds a new site containing the selected theme, he/she can send its URL to the WCA in order to start a deep search for useful information starting at that location.

In the on-line process, the WCA agent uses the last DSS sent by the Tutor, in order to classify and assign a trust factor to each page found.

The trust factor assigned is based on the performance estimated to the DSS in use by the WCA. After the classification process the pages are sent to the Miner agent.

2.5 The Miner Agent (MA)

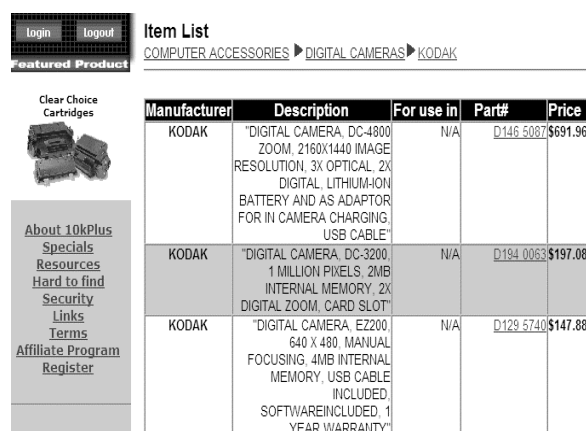
The MA is responsible for the analysis of the web pages found by the WCA. Once found and classified by the WCA, the MA analyses each page and the relevant information presented on the page will be selected for storage on the catalogue's database.

The main challenge for this agent is the selection of the data to be loaded into the catalogue. In fact, it is expectable to find a huge amount of non-significant data that must be avoided in order to produce easy-to-read and directly comparable catalogues. There are two fundamental problems to overcome, concept identification and concept classification.

To illustrate the problem under analysis, using a DEEPSIA's example, observe

Figure 2 where a web page for product selling is presented. Imagine what would be needed to identify the products included in the table. The first task is to identify how and where to find the product information (concept identification). The second task

would be the product's classification (concept classification).



Manufacturer	Description	For use in	Part#	Price
KODAK	"DIGITAL CAMERA, DC-4800 ZOOM, 2160X1440 IMAGE RESOLUTION, 3X OPTICAL, 2X DIGITAL, LITHIUM-ION BATTERY AND AS ADAPTOR FOR IN CAMERA CHARGING, USB CABLE"	N/A	D146.5087	\$691.96
KODAK	"DIGITAL CAMERA, DC-3200, 1 MILLION PIXELS, 2MB INTERNAL MEMORY, 2X DIGITAL ZOOM, CARD SLOT"	N/A	D194.0063	\$197.08
KODAK	"DIGITAL CAMERA, EZ200, 640 X 480, MANUAL FOCUSING, 4MB INTERNAL MEMORY, USB CABLE INCLUDED, SOFTWARE INCLUDED, 1 YEAR WARRANTY"	N/A	D129.5740	\$147.88

Figure 2: An example of a commercial page from the Internet

In the on-line process the MA uses a forward inference decision support system for concept identification and a reverse index keyword system for concept classification. The rules instantiated in both systems are maintained by the Tutor Agent.

Despite the classification given to each page by the WCA, the MA will overwrite this value according to the quality of the information found during the analysis process.

3 The Tutor Agent (TA)

The Tutor Agent's role is to support the user to perform all the text learning techniques, in an off-line process, in order to create Decision Support Systems (DSS) for the WCA and the MA.

3.1 Web Crawler Agent DSS

For the WCA, the TA's role is to support the user in performing the classical learning tasks of:

Corpus creation: the creation of a database of samples to perform supervised learning. As in all supervised learning process, knowledge is extracted from a set of classified observations (the corpus) that are previously stored (Mitchell, 1996). The corpus, used for the experimental results was created during the DEEPSIA project. The corpus was built using one hundred commercial web sites, from an arbitrary Internet site selection. The corpus included a total of 3579 documents, tagged as selling or non-selling

samples. Unlabelled documents, or double document tagging (i.e. one document classified in both categories) were not allowed. The corpus included 2528 selling documents and 1051 non-selling documents.

Feature selection: identification of the most relevant words to be selected for the vector in order to reduce the vector's size, usually superior to 30.000 dimensions (one dimension for each word found in the dictionary). The first step consisted in removing all the features included in the stop list DTIC/DROLS. This list retains most of the words in the standard Verity Stop List and adds the words from the DTIC/DROLS stop word list (the full list can be found at http://dvl.dtic.mil/stop_list.html). The second step consisted in performing feature selection (over all words still included in the corpus) using feature's Mutual Information (Yang and Pedersen, 1997), Conditional Mutual Information and Chi-square. Just the most expressive features are selected to be included in the vector for document representation. The exact number of features to be selected (the k-trash-older) is dynamically de-fined depending on the system's performance.

Creation of classifiers: the creation of several possible classifiers in order to produce alternative classification procedures. The classifiers are produced using the pre-selected features vector. The methods studied were the K nearest neighbour, K weighed – NN (Yang and Liu, 1999), C4.5 (Quinlan, 1993) and Bayesian classifier (Hastie, Tibshirani and Friedman, 2001).

Setting up a Decision Support System: the creation of rules in order to perform the classification task. The setting up of the DSS is defined based on the available classifiers and the analysis of their performance. The currently selected DSS is the rule of majority applied to the classifications assigned by the selected classifiers.

Creation of the Equivalent Decision Support System (EDSS) for Performance Optimisation. In order to enhance the system global performance an effort was made on the creation of an EDSS, which was presented in (Sousa, Pimentão, Pires and Garção, 2002).

Regarding feature selection methods, the best method used was the Conditional Mutual Information (CMI), because of its capability to eliminate correlated features. Genetic Algorithms were used to overcome the computational complexity with a

remarkable performance, even if they do not guarantee the best solution. The C4.5 classifier induction was used for testing the feature selection process. The slight improvements achieved with CMI over the other methods are particularly relevant, since C4.5 already eliminates correlated features.

Regarding the induced classifiers using the best feature ranking, the best results were achieved using decision trees.

Regarding the Decision Support System development the best results were achieved using nine C4.5 decision trees induced using different training sets over the original corpus. The trees were aggregated using the majority rule.

Finally, the created EDSS increased the system performance reducing the number of used decision trees.

4 Why Agent technology?

The agent technology is not a generic panacea to be applied to every day problems. Between others, the general complexity of the paradigm, the imposed communication and computational overhead, and the usage of a real distributed programming approach must be taken into account at the decision's moment. Furthermore, the lack of a sound theory and standards to support the analysis' and implementation process, together with the inexistence of robust, scalable and secure agent development platforms and agent deployment platforms are real obstacles to produce fast, reliable, and secure systems.

Therefore, the usage of agent's approach must be carefully analyzed and specially applied to real distributed, modular, decentralized decision making and ill-structured problems. Then the advantages will overcome the disadvantages and the achieved solutions are flexible, natural and elegant.

The nature of the problem under study is difficult to deal with, using traditional software engineering techniques. Even the traditional distributed approaches are difficult to applied because of the decentralized decision making and the geographical distribution. Therefore, the agent approach seemed suitable at the beginning of the project.

In fact, the agents' paradigm confirmed this assumption throughout the project execution coping efficiently with the day-by-day problems and situations.

The agents' modular approach enables a flexible source encapsulation increasing reusability, and fast and easy error detection and correction. The fact that every agent implements an independent decision maker allowed the implementation of a natural decentralized decision system. The straightforward capability to introduce, eliminate, or change agents, or agent types, adapted to the problem's ill-structure facilitated the development and implementation, since the error reflexes, or unexpected behaviors resulting from changes are self-contained and avoided secondary reflexes.

The agents' intelligence is the result of applying learning techniques, specifically the supervised text learning and the if-then-else rules.

5 Conclusions

The "Framework for Internet data collection based on intelligent agents" is a generic approach; its context definition is done using the training process of the Crawler and of the Miner agents. Depending on the class, the ontology and the rules definition, the system may be adapted to distinctive objectives and we hope time and experience will continue to support our expectations.

The framework was under extensive tests in the DEEPSIA's project, and the achieved results were very positive and above the consortium's expectations. Although the used corpus is dedicated to e-procurement, the global results achieved are encouraging and the framework is now under testing with other corpus.

The use of different learning techniques and DSS was a priority from the beginning of the project as the joint effort of all learning techniques increases drastically the performance of the platform, taking advantage of the particularities of each technique concerning its success ratio in terms of quantity and aim.

The Framework has shown to be fit for the application in the DEEPSIA scenario. During the project, we reached the conclusion that SMEs, would rather prefer to monitor a set of pre-selected sites (their usual suppliers), than to search the whole World-Wide-Web trying to identify new web sites (most of them not relevant to the business of the company). In this environment the DEEPSIA solution is quite effective. Since the corpus from where the DSSs are inducted is created with the pages from the

user's selected sites, the corpus is roughly equal to the universe; therefore the estimated precision and recall values achieved are very similar to the real values. For further information about the project, the latest achievements and access to the prototype please consult the DEEPSIA's web site on www.deepsia.org.

Generically speaking, the architecture contributes to a new solution for information retrieval based on text learning techniques.

The usage of agents was fundamental to achieve the results in a short period.

The association of both technologies, agents and leaning, enabled promising results and the creation of a generic approach applicable in different environments. Its usage in Finance and Military fields are already under analysis with similar results.

6 References

- P. A. C. Sousa, J. P. Pimentão, F. M. Pires and A. Garção. A framework for Internet data collection based on intelligent Agents: The methodology to produce equivalent DSS. *Smart Engineering System Design: Neural Networks, Fuzzy logic, Evolutionary programming, data mining, and complex systems*, St. Louis, USA, ASME, 147-151, 2002
- T. Tan. *The development of Intelligent Conceptual storage and retrieval system*. Sunderland, UK, University of Sunderland, 1993
- ECCMA. *ECCMA web site*. 2002
- FIPA00061. *FIPA ACL Message Structure Specification*. 2000
- T. M. Mitchell. *Machine Learning*. McGraw-Hill International Editions, 1-414, 1996
- Y. Yang and J. P. Pedersen. A Comparative Study on Feature Selection in Text Categorization. *ICML'97 - Fourteenth International Conference on Machine Learning*, 412-420, 1997
- Y. Yang and X. Liu. A re-examination of text categorization methods. *SIGIR'99*, 1999
- J. Quinlan. *C4.5: Programs for Machine Learning*. San Mateo, Morgan Kaufmann, 1993
- T. Hastie, R. Tibshirani and J. Friedman. *The elements of statistic learning, Data Mining, Inference, and prediction*. Springer, 2001

Automated Component-Based Configuration: Promises and Fallacies

Sander van Splunter*

Niek Wijngaards*

Frances Brazier*

Debbie Richards†

*Department of Computer Science
Vrije Universiteit Amsterdam, The Netherlands
{svsplun,niek,frances}@cs.vu.nl

†Department of Computing, Macquarie University,
Sydney, Australia
richards@ics.mq.edu.au

Abstract

Re-use of software components is standard practice in software design and development in which humans play an important role. In many dynamic environments, however, (semi-)automated configuration of systems, is warranted. This paper examines three such domains: Agent Factories, Web service configuration and general software composition. The differences and similarities between these approaches, and the progress that is being made are discussed.

1 Introduction

Re-use of software components is part of many approaches to software design and development ((e.g., Biggerstaff and Perlis (1997))). Most approaches assign an important role to human developers. In dynamic environments (semi-)automated configuration of systems can reduce, or even eliminate, the human effort required. In dynamic environments (semi-)automated configuration of systems from reusable components, is warranted. This paper examines three such domains: Agent Factories, Web service configuration and general software composition. The differences and similarities between these approaches, and the progress that is being made, are explicitly addressed.

Agents are active entities in dynamic, changing environments supported by the Internet. There are different ways for agents to adapt to such changing environments (e.g., see Splunter et al. (2003)). One is the Agent Factory approach in which first a need for change is identified and then agents are adapted. Another is the evolutionary approach in which agents continually adapt to their environment through implicit learning. The first mandates an understanding of the structure of an agent, and the components involved. The second mandates an understanding of the parameters involved in learning. This paper examines the first approach.

The Internet provides infrastructure to host agents and mobile processes. It also provides the infrastructure needed for both agents and humans to access Web services. In many business chains a number of Web services play a role. Web services need to be combined - configured. Availability of Web services, for example, is often crucial. If a particular Web service is not accessible an

alternative service or combination of services needs to be found almost instantaneously. Automated configuration, although not currently acquired, is being actively pursued.

Within component based software engineering approaches automatic configuration is not often pursued, but, when it is, it focuses mostly on the initial design. It, thus, provides the basis for both agent and Web service configuration.

This paper is organised as follows. Section 2 briefly introduces and analyses three Agent Factories. Section 3 introduces and analyses Web service configuration. Section 4 introduces and analyses one approach to component-based configuration from a software engineering perspective. Section 5 compares these three overall approaches to component-based configuration by focussing on strengths and weaknesses on a number of aspects, related to the end-products, reusable components, and the configuration process. Section 6 concludes this paper with directions for future research.

2 Agent Factories

Agent Factories are either services or toolkits for (semi-)automated creation and (optionally) adaptation of software agents. These environments are strongly related to methodologies for agent application development. They include support for agent modelling (e.g. AURL), generic agent models (e.g. by DESIRE, ZEUS, or InterRap) and prototype generation. Examples of prototyping environments include the ZEUS toolkit (Nwana et al. (1998)), LEAP (Berger et al. (2001)), the (Dutch) Agent Factory (Brazier and Wijngaards (2001)), the (Irish) Agent Factory (Collier and O'Hare (1999)), and the (Italian) Agent Factory (Cossentino et al. (2003)). This paper

refers to the last three as the Dutch, Irish, and Italian agent factories and describes them in more detail in this paper.

In general, agent factories produce software agents, which are executed in the context of specific agent platforms.

Dutch Agent Factory. This approach focuses on automation of the creation and adaptation of compositional agents. The nature of the components, of which an agent is composed, is graybox. These components provide mechanisms to implement an agent's processes, knowledge & information and control. Component composition is regulated by explicitly defined 'open slots' in components & templates based on generic agent models. Components are defined at two levels of abstraction: conceptual and operational. Minimal ontologies are used for annotation of components and interfaces, without adhering to standard ontologies or languages. Repositories for building blocks have not yet been further researched; only local re-use is supported.

The configuration process itself is automated, and includes reasoning on, and modification of, the requirements on the desired configuration. Configuration creation is automated, based on a generic model of design theory: the Generic Design Model (Brazier and Wijnngaards (2002)). Retrieval and assembly are modelled as separate processes. Adaptation is approached as re-design, supported by the Generic Design Model. Component retrieval has not yet been automated. Assembly has been automated, partially being incorporated in the operational architecture used for the agents. Execution of configured agents is done by means of an agent platform. Agent platforms for which agents have been configured are AgentScape, FIPA-OS, and JADE.

Italian Agent Factory. The Italian Agent Factory is based on a toolkit that supports the multi-agent system design methodology PASSI (Cossentino and Potts (2002)).

The nature of the components is graybox. Components are configured and structured for five different configuration perspectives used within PASSI: knowledge, social, computer, architectural, and resource. The first two can be related to the conceptual level, and the remaining three to the operational level. Conceptual components are based on roles and tasks, operationally software component are used for modelling. Component composition is done by pattern merging on the conceptual level, on the operational level code is reused linked to the conceptual patterns. The interfaces of the software components are locally developed and defined.

The configuration process results in a skeleton of code that needs to be completed by a human programmer. Annotation of the different patterns is done using AUMIL, state charts, and activity diagrams. Annotation for human designers is supported by the Rational Rose tool. The creation of an extensive repository of patterns is still an open

research issue. The re-use community of the patterns is limited to the Italian Agent Factory.

The Italian Agent Factory's configuration creation process is semi-automated as a support tool. Configuration adaptation does not seem to be explicitly supported. Pattern retrieval seems to be mainly done by a human designer. The assembly of components is partially automated: a skeleton with partial completed code is the software product of the Italian Agent Factory. The execution is by means of a FIPA-compliant agent platform. The agents produced adhere to the interface standards set by FIPA for agent platforms.

Irish Agent Factory. The Irish Agent Factory is developed as a complete system to enable Agent Oriented Software Engineering, complete with a formal theory on agent commitments, agent programming language, and a run-time environment. This paper focuses on the tool to create agents.

The Irish software components are graybox, all programmed and developed within the same methodology. The conceptual components are modeled as roles and tasks, time with a focus on the use of commitments. The components' interfaces are developed and defined locally. Conceptually component composition is done by configuring a set of actuators and perceptors. Operationally agents are created by module-based development. Different default sets of actuators, perceptors, and modules are offered to support default implementations of different classes of agents. Annotation is application specific in the Irish Agent Factory's own high level programming language (AF-APL) with the addition of behaviour diagrams. Repositories of components have not been developed. Due to the specific theory, programming language, and run-time environment the reuse community consists only of the Irish Agent Factory.

Configuration creation is primarily done by a human designer, where the toolkit mainly acts as a smart interface. A number of default configurations is offered that can be used for initial configuration. Component retrieval is limited, a lookup service is associated which can (partial) retrieve designs via design identifiers. Assembly is done by either retrieving pre-fabricated configurations, or semi-automated by the toolkit. Execution is by means of the agent platform associated with the Irish Agent Factory.

Discussion. In the three Agent Factories discussed above agents are developed on the basis of instantiated 'patterns' (e.g. 'generic models'), or combinations of agent-components. The Italian and Irish agent factories focus on semi-automated generation; automated generation of agents is demonstrated by the Dutch agent factory. Automated adaptation of previously configured agents is only achieved by the Dutch Agent Factory. Nevertheless, all three Agent Factories pave the road for component-based agent *adaptation*.

All these Agent Factories produce agents for agent platforms that are FIPA compliant. Each agent factory uses its own approach to define components, interfaces, and annotation. All agent factories provide mechanisms related to process, data/information & knowledge, and control within software agents. All agent factories distinguish conceptual and operational levels of configuration. At the conceptual level, the Italian agent factory merges its components (patterns), while the Dutch and Irish agent factories combine components. All agent factories combine components at the operational level.

All agent factories are rather small-scale, both in the number of annotated components provided and associated (re-use) community. Annotation of components is often not explicitly supported, but sometimes implicit in the development of components (i.e. the Irish Agent Factory).

Not all agent factories use standard Software Engineering modelling support for creating agents. The Italians use AUML, state charts, activity diagrams, and extended Rational Rose. The Irish have explored the option of extending their methodology with AUML. The Dutch Agent Factory lacks standard SE modelling support technologies.

3 Web Service Configuration

The Stencil Group¹ defines web services as "loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols". Web services are related to the Semantic Web, in which data is defined and linked to make it accessible and interpretable for automated systems.

A configuration of Web services describes which Web services combined, control, and the information exchange. A configuration of Web services involves multiple processes on different hosts as individual Web services are offered and hosted by different parties. In contrast to agents, the behaviour of both a configuration and of single Web services is purely reactive and static. In general, two perspectives on Web service configuration can be identified (Srivastava and Koehler (2003)): a more syntactic-oriented Business Process composition and a more Semantic-Oriented service composition. The Business Process perspective models Web services as business processes, without attaching detailed semantics, and uses standardised technologies to describe Web services such as WSDL, SOAP, and UDDI. Web service configurations are described by orchestration languages such as XLANG, WFSL, BP4WS, or WSMF. This perspective is successful in human-supported discovery, composition, and monitoring of Web services. The Semantic-Oriented perspective extends the Business Process perspective by identifying the need of explicit semantics to enable the automa-

tion these tasks. Standard languages (e.g. OWL) and ontologies (e.g., OWL-S (formerly DAML-S)) are available for semantically annotating Web services. This paper focuses on the two Semantic-Oriented approach discussed below.

eFlow. eFlow (Casati et al. (2000)), developed by the Hewlett-Packard Company, is oriented to adaptation of workflow models of composite Web services. Web services are treated as blackbox components. The component structure is based on workflows: activities with a data flow and control flow are modelled. The component interfaces are based on standard Internet protocols, and are described in WSDL. Component composition is regulated using a self-defined model, in which workflow concepts have been reused and extended. An XML specification and a DTD to constrain syntax are used for annotation. The component availability is low, due to the specific XML annotation.

eFlows main consideration is the focus on adaptation. Adaptations is performed in the context of monitoring Web service configuration, to minimize or eliminate human intervention. Component retrieval is done by brokers using centralised repositories, and the execution and monitoring is performed by an eFlow engine.

Cardoso Cardoso and Sheth (2002) focus on the integration of new Web services in existing workflows. Web services are blackbox components. The component structure is based on modelling activities with a data flow and control flow, extended with annotations on the Quality of Service (QoS). Component composition is based on workflow integration, and supported by abstract Service Templates. For annotation DAML-S has been used in examples. The DAML-S Profile ontology has been extended to include more details on the QoS. This approach has a prototype with a self-defined local repository and discovery service, though usage of UDDI is also considered. The reuse community of the components is larger (the additional QoS attributes are only extensions to existing standards, not rendering components incompatible with other approaches). Component availability, including QoS annotations, is small.

In the configuration process of Cardoso's approach configuration adaptation is semi-automated. The human designer is supported in the creation and refinement of abstract Service Templates. This approach is not specifically targeted to the creation of Web service configurations from scratch. Component retrieval is semi-automated: based on an abstract Service Template possible Web services for refinements are retrieved based on aspects of QoS, similarity of textual descriptions or names, and semantic similarity of inputs and outputs. The assembly and execution a configuration is not clear.

Discussion. Within Web services configuration the services are blackbox components with standardised inter-

¹http://www.stencilgroup.com/ideas_scope_200106wsdefined.html

faces. The component interface standards are based on standard Internet protocols and described using WSDL and SOAP: generally accepted standards. The component structure is mostly process-based: in the workflow perspective it consists of activities with a control and data flow.

The components themselves are annotated in standard annotation languages as WSDL and OWL. The latter language is mostly used by semantic-oriented matching approaches, for which standard ontologies for Web services are available in OWL-S. Annotating Web services is an inherent part of the Web service development process. Unfortunately, most approaches do not focus on creating rich domain ontologies to be reused when describing other services. A large number of Web services is available where annotations are limited to only the WSDL descriptions. A smaller number of components is available with semantic rich OWL-S descriptions

The use of globally shared repositories support a global reuse community and widely available Web services. The communities involved in the semantic Web are still growing, which may positively influence the availability of well-annotated re-usable Web services.

As Web services, by their very nature, can appear, change, or disappear while being used in compositions, the need for automated adaptation is recognised, and progress is made. Most approaches to automated configuration are still in development; configuration creation is often approached as a (simple) planning problem. Component retrieval is semi-automated, i.e. UDDI is used as a repository that can be queried, but human intervention is still required to determine whether the resulting Web service are useable. The Semantic-Oriented perspective has extended (e.g. MatchMaker by Paolucci et al. (2002)) UDDI to handle further automated semantic querying, by including approximate answers. Execution of Web services configurations is done by workflow execution engines, e.g., BPWS4J, and the BPEL Orchestration Server. For execution of DAML-S descriptions research is still evolving (e.g. see Gaio et al. (2003)), as the research area is still young.

4 Component-Based Software Engineering

Component-based software engineering focuses, in general, on developing components (e.g., CORBA, Java Beans, .NET). while software composition is often only supported by tools, not automated, although exceptions are present. Examples of modelling support are UML², or tools such as Rational Rose³. In this paper Quasar(de Bruin and van Vliet (2003)), a semi-automated approach to software composition based on feature composition, is discussed.

²<http://www.uml.org>

³<http://www.rational.com>

Quasar. Quasar is a tool to support top-down composition of software architectures. In this approach, an architecture is derived to fulfill a set of quality concerns. A Quasar specific feature-solution graph is used to connect quality requirements to solution fragments at the architectural level: a form of composition knowledge. An architecture is derived by systematically composing solution fragments. Both functional and non-functional requirements are addressed.

The reusable components, in this approach design solutions, are often patterns but may also be individual software components. Design solutions are represented by use case maps (UCM) whereby both behavioural and structural aspects are expressed. Components are distinguished from sockets and stubs, with which component composition is regulated, via a refinement process. Quasar uses BCOOPL(de Bruin (2003)) to specify interface definitions, including pre- and post-conditions and -processing via pre- and post-stubs. Composition templates support composition of design solutions.

Annotation of design solutions are encoded in feature-solution graphs. These graph are not automatically generated. A number of prototype components are available. The configuration creation process is semi-automated: architectures are derived iteratively by first generating a reference architecture, focussing on functional requirements followed by non-functional requirements. The choices of which requirements to focus on, is left to the human designer. This process may include backtracking to resolve conflicting requirements.

Discussion. Component-based software configuration is a broad field. Automated approaches in component-based software engineering are often very domain-specific (comparable with agent-configuration and Web service configuration). In general, components are well-defined structures, for which composition is explicitly regulated by defining 'hooks'. Both components and hooks are well-defined on a syntactical level, sometimes involving semantic annotations (by associating features to solutions in Quasar). Only small sets of annotated components are available, and the more general an approach, the more support is provided for large-scale repositories. Simple, exact-matching, retrieval is often provided. Configuration of software components is almost never fully automated. This is due, in general, to software components being difficult to reuse as their domain specificity conflicts with reuse genericity (Sametinger (1997)). Assembly of software compositions is, in general, supported.

5 Comparative Analysis

Agents, services and general compositional software have different characteristics, but also commonalities. The comparison in this section is structured by focussing on

four aspects: component definition, component annotation, component availability, and configuration process.

Component definition. Although all approaches explicitly define components, the Web service approaches employ standards, used by a large community. The component definition, however, is a blackbox approach, without providing hooks for composition: composition requires new configuration languages. In the Agent Factories and component-based software engineering components are graybox, providing hooks for composition. All approaches distinguish conceptual and operational levels in their component definitions.

Web services are intended to be a globally reused, in an open domain. The Web service community builds on the existing (Web-)protocols, e.g. SOAP, HTTP, while the agent community has inter-agent protocols, but have no standards for the interfaces of the components of which agents themselves are composed. The software composition approach provide no restrictions on domain and/or interfaces, yet automated configuration is limited to specific domains of application.

Component annotation. The adoption of standards for annotation facilitates the development of automated configuration processes. Only the Web service community has emerging computer-interpretable annotation standards, the other approaches do not as yet. Human-understandable annotations of software engineering, like UML, are widely applied and accepted (broader than the Web service annotation standards). The annotations of the Web service communities do not extend these SE standards. Within the Agent modelling community the SE annotations are more generally accepted (e.g. AUML).

Component availability. Automated configuration of software depends on the availability of configurable software components: a critical mass is needed for automated configuration to be successful. Annotated Web services are becoming more widely available, due to a large supportive community and the creation of semantic descriptions being part of the development process of Web services. Agents, however, are often developed without a focus on re-usability of agent-components, leading to scarce availability of annotated agent-components. Software components are being developed in large quantities, around the world, but are usually not geared for automated reuse, lacking computer-interpretable annotations.

Configuration processes. Fully automated configuration is not widely supported. All communities research automated adaptation, whereby the agent community often focuses on learning algorithms without structural adaptation. The Web service community focuses on adaptation, mostly studied in the context of business-processes. Component-based software engineering has

traditionally had its main focus semi-automated adaptation, but is moving towards automated adaptation (e.g., see also the developments in self-managing, and self-healing systems).

Discovery and retrieval of software components is important. Annotation of components will facilitate their automatic discovery and retrieval. The Semantic Web community experiments with public services, and uses reasoning about the annotations. Within the Agent Factories retrieval of re-usable components is not extensively studied, and no large repositories are publicly available. For software-composition communities repositories are available, however mainly for computer-supported discovery and retrieval. For automated discovery and retrieval the repositories are often limited in size and non-public.

6 Discussion & Future Work

The comparison is limited, discusses the issues involved in each of the approaches. Internet applications require flexibility to accomodate changes in their environment. As manual adaptation is not pragmatic when multitudes of agents and Web services are in use, automated adaptation becomes a necessity. Unfortunately, the current state of the art, even in software engineering, does not include fully automated component-based adaptation. Current research focuses on component-based configuration of agents, Web services, and software composition: a precondition for adaptation. Progress has been made in automation of component-based configuration.

Automated component-based configuration of, e.g., software agents, entails a thorough understanding of both configuration processes, and the components to be configured. The agent community has made the most progress in automation of the configuration process. The Web service community has made the most progress in development and annotation and in the discovery and retrieval of these components. The software composition community has made the most progress in structuring and modelling components, and on supporting the human designer. Interdisciplinary research may be most fruitful, when (1) combining configuration-expertise with annotation-expertise, (2) generalising and standardising reusable, configurable, components, and (3) when the current structuring and modelling practices of SE are used. Once automated component-based configuration has proven to be feasible, research can focus on automated component-based adaptation as a new challenge.

Acknowledgements

The authors wish to thank Marta Sabou for her work on Web service configuration. The authors are also grateful to the support provided by Stichting NLnet, <http://www.nlnet.nl/>.

References

- N. Berger, B. Bauer, and M. Watzke. A scalable agent infrastructure. In *2nd Workshop on Infrastructure for Agents, MAS and Scalable MAS. Autonomous Agents.01, Montreal*, 2001.
- T. Biggerstaff and A. Perlis, editors. *Software Reusability: Concepts and models*, volume 1. ACM Press, New York, 1997.
- F.M.T. Brazier and N.J.E. Wijnngaards. Automated servicing of agents. *AISB Journal*, 1(1):5–20, 2001. Special Issue on Agent Technology.
- F.M.T. Brazier and N.J.E. Wijnngaards. Automated (re-)design of software agents. In J.S. Gero, editor, *Proceedings of the Artificial Intelligence in Design Conference 2002*, pages 503–520. Kluwer Academic Publishers, 2002.
- J. Cardoso and A. Sheth. Semantic e-workflow composition. Technical report, LSDIS Lab, Computer Science Department, University of Georgia, 2002. <http://chief.cs.uga.edu/~jam/webwork/geneflow/papers/CS02-20Composition.20-20TR.pdf>.
- F. Casati, S. Ilnicki, and L. Jin. Adaptive and dynamic service composition in eflow. HP Technical Report HPL-2000-39, HP, 2000. <http://www.hpl.hp.com/techreports/2000/HPL-2000-39.pdf>.
- R.W. Collier and G.M.P. O'Hare. Agent factory: A revised agent prototyping environment. In *10th Irish Conference on Artificial Intelligence & Cognitive Science*, University College Cork, Ireland, 1999.
- M. Cossentino, P. Burrafato, S. Lombardo, and L. Sabatucci. Introducing pattern reuse in the design of multi-agent systems. In R. Kowalczyk, J.P. Muller, H. Tianfield, and R. Unland, editors, *Agent Technologies, Infrastructures, Tools, and Applications for E-Services: NODE 2002 Agent-Related Workshops, Erfurt, Germany, October 7-10, 2002.*, volume 2592 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 107–120, 2003.
- M. Cossentino and C. Potts. A case tool supported methodology for the design of multiagent systems. In *Proceedings of the 2002 International Conference on Software Engineering Research and Practice (SERP'02)*, Las Vegas, NV, USA, 2002.
- H. de Bruin. Bcoopl: A language for controlling component interactions. *The Journal of Supercomputing*, 24(2):131–139, 2003.
- H. de Bruin and H. van Vliet. Quality-driven software architecture composition. *Journal of Systems and Software*, 66(3):269–284, 2003.
- S. Gaio, A. Lopes, and L. Botelho. From daml-s to executable code. In B. Burg, J. Dale, T. Finin, H. Nakashima, Padgham L., C. Sierra, and Willmott S., editors, *Agentcities: Challenges in Open Agent Environments*, pages 25–31. Springer-Verlag, 2003.
- H.S. Nwana, D.T. Ndumu, and L.C. Lee. Zeus: An advanced tool-kit for engineering distributed multi-agent systems. *Applied AI*, 13(1/2):129–185, 1998.
- M. Paolucci, T. Kawamura, T.R. Payne, and K.P. Sycara. Semantic matching of web services capabilities. In *Proceedings of the International Semantic Web Conference 2002*, pages 333–347, 2002.
- J. Sametinger. *Software engineering with reusable components*. Springer Verlag, New York, 1997.
- S. van Splunter, N.J.E. Wijnngaards, and F.M.T. Brazier. Structuring agents for adaptation. In E. Alonso, D. Kudenko, and D. Kazakov, editors, *Adaptive Agents and Multi-Agent Systems*, volume 2636 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 174–186. Springer-Verlag Berlin, 2003.
- B. Srivastava and J. Koehler. Web service composition - current solutions and open problems. In *ICAPS 2003 Workshop on Planning for Web Services*, pages 28–35, 2003.