AISB'02 Convention

3rd-5th April 2002

Imperial College

Proceedings of the AISB'02 Symposium on Adaptive Agents and Multi-Agent Systems

Contents

Symposium Preface
Adaptive Agents and Multi-Agent Systems (AAMAS-II)

The Symposium on Adaptive Agents and Multi-Agent Systems (AAMAS-II) is a continuation of AAMAS. AAMAS was held as part of AISB'01 in York, March 2001. AAMAS was a pioneering experience, as no symposium on learning agents had been organised previously in the UK. The success of the symposium has encouraged us to organise AAMAS-II.

The goals of the symposium are as follows:

- Increase awareness and interest in adaptive agent research in the AI community and encourage further research
- Encourage collaboration between ML experts and Agent system experts
- Give a representative overview of current research in the area of adaptive agents in Europe and world-wide

We sincerely thank our invited speaker, Saso Dzeroski, and our programme committee, Kurt Driessens, Peter Edwards, Eugenio Oliveria, Michael Schroeder, Kostas Stathis, and Niek Wijngaards, and other reviewers, Chris Child, David Mobach, Ana Paula Rocha, and Sander van Splunter.
We would like to thank the AgentLink2 network for their financial support.

Eduardo Alonso, Daniel Kudenko and Dimitar Kazakov

Relational Reinforcement Learning for Agents in Worlds with Objects

Sašo Džeroski
Institut Jožef Stefan
Jamova 39, SI-1000 Ljubljana, Slovenia
Saso.Dzeroski@ijs.si

Abstract

In reinforcement learning, an agent tries to learn a policy, i.e., how to select an action in a given state of the environment, so that it maximizes the total amount of reward it receives when interacting with the environment. We argue that a relational representation of states is natural and useful when the environment is complex and involves many inter-related objects. Relational reinforcement learning works on such relational representations and can be used to approach problems that are currently out of reach for classical reinforcement learning approaches.

1 Introduction

In reinforcement learning, an agent tries to learn a policy, i.e., how to select an action in a given state of the environment, so that it maximizes the total amount of reward it receives when interacting with the environment. In cases where the environment is complex and involves many inter-related objects, a relational representation for states is natural. This typically yields a very high numbers of possible states and state/action pairs, which make most of the existing tabular reinforcement learning algorithms inapplicable. Even the existing reinforcement learning approaches that are based on generalization typically use a propositional representation and cannot deal directly with relationally represented states.

We introduce relational reinforcement learning, which uses relational learning algorithms as generalization engines within reinforcement learning. We start with an overview of reinforcement learning ideas relevant to relational reinforcement learning. We then introduce several complex worlds with objects, for which a relational representation of states is natural. An overview of different relational reinforcement learning algorithm developed over the last five years is presented next and illustrated on an example from the blocks world. Finally, some experimental results are presented before concluding with a brief discussion.

2 Reinforcement Learning

This section gives an overview of reinforcement learning ideas relevant to relational reinforcement learning. For an extensive treatise on reinforcement learning, we refer the reader to (Sutton & Barto, 1998). We first state the task of reinforcement learning, then briefly describe Q-learning. In its basic variant, Q-learning is tabular: this is unsuitable for problems with large state spaces, where generalization is needed. We next discuss generalization in reinforcement learning and in particular generalization based on decision trees. Finally, we discuss the possibility of integrating learning by exploration (as is typically the case in reinforcement learning) and learning from guidance (by a human operator or some other reasonable policy).

2.1 Task definition

The typical reinforcement learning task using discounted rewards can be formulated as follows:

Given

- a set of possible states S.
- a set of possible actions A.
- an **unknown** transition function δ: $S \times A \rightarrow S$.
- an **unknown** real-valued reward function $r : S \times A \rightarrow R$.

Find a policy $\pi^* : S \rightarrow A$ that maximizes

$$V^{\pi}(s_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

for all s_t where $0 \leq \gamma < 1$.

At each point in time, the reinforcement learning agent can be in one of the states s_t of S and selects an action $a_t = \pi(s_t) \in A$ to execute according to its policy π. Executing an action a_t in a state s_t will put the agent in a new state $s_{t+1} = \delta(s_t, a_t)$. The agent also receives a reward $r_t = r(s_t, a_t)$. $V^{\pi}(s)$ denotes the value (expected return; discounted cumulative reword) of state s under policy π.

Table 1: The Q-learning algorithm.

```
Initialize Q(s, a) arbitrarily
Repeat (for each episode)
    Initialize s
    Repeat (for each step of episode)
        Choose a from s using policy derived from Q
        Take action a, observe r, s'
        Q(s, a) ← r + γ max_a' Q(s', a')
        s ← s'
    Until s is terminal
```

It will be assumed that the agent does not know the effects of the actions, i.e., δ is unknown to the agent, and that the agent does not know the reward function r. The task of learning is then to find an optimal policy, i.e., a policy that will maximize the discounted sum of the rewards. We will assume episodic learning, where a sequence of actions ends in a terminal state.

2.2 Tabular Q-learning

Here we summarize Q-learning, one of the most common approaches to reinforcement learning, which assigns values to state-action pairs and thus implicitly represents policies. The optimal policy π^* will always select the action that maximizes the sum of the immediate reward and the value of the immediate successor state, i.e.,

$$\pi^*(s) = argmax_a(r(s,a) + \gamma V^{\pi^*}(\delta(s,a)))$$

The Q-function for policy π is defined as follows :

$$Q^\pi(s,a) = r(s,a) + \gamma V^\pi(\delta(s,a))$$

Knowing Q^*, the Q-function for the optimal policy, allows us to rewrite the definition of π^* as follows

$$\pi^*(s) = argmax_a Q^*(s,a)$$

An approximation to the Q-function, $\hat{Q}$, in the form of a look-up table, is learned by the following algorithm.

The agent learns through continuous interaction with the environment, during which it exploits what it has learned so far, but also explores. In practice, this means that the current approximation Q is used to select an action most of the time. However, in a small fraction of cases an action is selected randomly from the available choices, so as to explore and evaluate unseen state/action pairs.

For smoother learning, an update of the form $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_{a'} Q(s',a') - Q(s,a)]$ would be used. This is a special case of temporal-difference learning, where algorithms such as SARSA also belong. In SARSA, instead of taking the maximum over possible actions, the action a' is also chosen according to the same policy and the following update rule is used $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$. For the algorithm in Table 1, the learned action-value function Q directly approximates Q^*, regardless of the policy being followed.

2.3 Generalization / G-trees

Using a tabular representation for the learned approximation to the Q-function or V-functions is only feasible for tasks with small numbers of states and actions. This due to both space problems (large table), but also time problems (time needed to fill the table accurately). The way out is to generalize over sates and actions, so that approximations can be produced also for states (and possibly actions) that the agent has never seen before.

Most approaches to generalization in reinforcement learning use neural networks for function approximation (Bertsekas & Tsitsiklis, 1996). States are represented by feature vectors. Updates to state-values or state-action values are treated as training examples for supervised learning. Nearest-neighbor methods have been also used, especially in the context of continuous states and actions (Smart & Kaelbling, 2000).

Table 2: The G-algorithm.

```
Create an empty leaf
While data available do
    Sort data down to leaves
    Update statistics in leaves
    If a split is needed in a leaf,
        then grow two empty leaves
```

The G-algorithm (Chapman & Kaelbling, 1991) is a decision tree learning algorithm that updates its theory incrementally as examples are added. An important feature is that examples can be discarded after they are processed. This avoids using a huge amount of memory to store examples. At a high level, the G-algorithm (Table 2) stores the current decision tree, and for each leaf node statistics for all tests that could be used to split that leaf further. Each time an example is inserted, it is sorted down the decision tree according to the tests in the internal nodes; in the leaves, the statistics of the tests are updated.

2.4 Exploration and guidance

Besides the problems with tabular Q-learning, large state/action spaces entail another type of problem for reinforcement learning. Namely, in a large state/action space, rewards may be so sparse that with random exploration (as is typical at the start of a reinforcement learning run) they will only be discovered extremely slowly. This problem has only recently been addressed for the case of continuous state/action spaces.

(Smart & Kaelbling, 2000) integrate exploration in the style of reinforcement learning with human-provided guidance. Traces of human-operator performance are provided to a robot learning to navigate as a supplement to its reinforcement learning capabilities. Using nearest-neighbor methods together with precautions to avoid overgeneralization, (Smart & Kaelbling, 2000) show that using the extra guidance helps improve the performance of reinforcement learning.

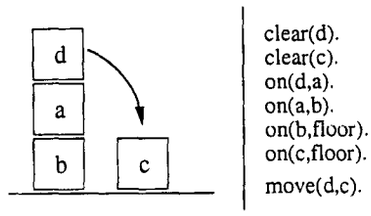


Figure 1: Example state and action in the blocks-world.

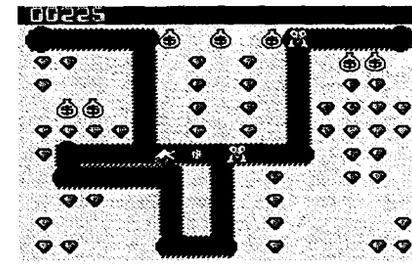


Figure 2: A snapshot of the DIGGER Game.

3 Some Worlds with Objects

In this section, we introduce three domains where using a relational representation of states is natural and involves objects and relations between them. The number of possible states in all three domains is very large. The three domains are: the blocks world, the Digger computer game, and the Tetris computer game.

3.1 The blocks world

In the blocks world, blocks can be on the floor or can be stacked on each other. Each state can be described by a set (list) of facts, e.g., $s = \{clear(c), clear(d), on(d,a), on(a,b), on(b,floor), on(c,floor)\}$ represents the state in Figure 1. The available actions are then $move(X,Y)$ where $X \neq Y$, X is a block and Y is a block or the floor. The number of states in the blocks world grows very fast with the number of blocks. With 10 blocks, there are close to 59 million possible states.

We study three different goals in the blocks world: stacking all blocks, unstacking all blocks (i.e., putting all blocks on the floor) and putting a specific block on top of another specific block.

In a blocks world with 10 blocks, there are 3.5 million states which satisfy the stacking goal, 1.5 million states that satisfy a specific on(A,B) goal and one state only that satisfies the unstacking goal. A reward of 1 is given in case a goal-state is reached in the optimal number of steps; the episode ends with a reward of 0 if it isn't.

3.2 The Digger game

Digger[1] is a computer game created in 1983, by Windmill Software. It is one of the few old computer-games which still hold a fair amount of popularity. In this game, the player controls a digging machine or "Digger" in an environment that contains emeralds, bags of gold, two kinds of monsters (nobbins and hobbins) and tunnels. The object of the game is to collect as many emeralds and gold as possible while avoiding or shooting monsters.

In our tests we removed the hobbins and the bags of gold from the game. Hobbins are more dangerous than nobbins for human players because they can dig their own tunnels and reach Digger faster as well as increase the mobility of the nobbins. However, they are less interesting for learning purposes, because they reduce the implicit penalty for digging new tunnels (and thereby increasing the mobility of the monsters) when trying to reach certain rewards. The bags of gold we removed to reduce the complexity of the game.

A state representation consists of the following components: coordinates of digger (e.g., $digPos(6,9)$), information on digger (of the form `digInf(digger_dead, time_to_reload, level_done, pts_scored, steps_taken)`, e.g., $digInf(false, 63, false, 0, 17)$, tunnels as seen by digger (range of view in each direction, e.g., $tunnel(4,0,2,0)$), list of emeralds (e.g., $[em(14,9), em(14,8), em(14,5), \ldots]$), list of monsters (e.g., $[mon(10,1,down), mon(10,9,down)]$), and information on the fireball fired by the digger (coordinates, travelling direction, e.g., $fb(7,9,right)$). The actions are of the form $moveOne(X)$ and $shoot(Y)$, where X and Y are in $[up, down, left, right]$.

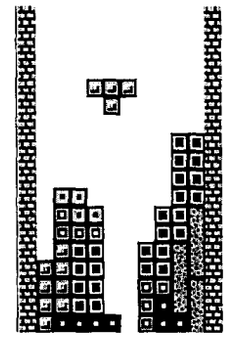


Figure 3: A snapshot of the TETRIS Game.

3.3 The Tetris game

Tetris[2] is a widespread puzzle-video game played on a two-dimensional grid. Differently shaped blocks fall from the top of the game field and fill up the grid. The object of the game is to keep the blocks from piling up to the top of the game field. To do this, one can move the dropping blocks right and left or rotate them as they fall. When one horizontal row is completely filled, that line disappears and the player scores points. When the blocks pile up to the top of the game field, the game ends.

[1] http://www.digger.org

[2] Tetris was invented by Alexey Pazhitnov and is owned by The Tetris Company and Blue Planet Software.

In the tests presented, we only looked at the strategic part of the game, i.e., given the shape of the dropping and the next block, one has to decide on the optimal orientation and location of the block in the game-field. (Using low level actions — turn, move left or move right — to reach such a subgoal is rather trivial and can easily be learned by (relational) reinforcement learning.) We represent the full state of the Tetris Game, the type of the next dropping block included.

4 Relational Reinforcement Learning

Relational reinforcement learning (RRL) addresses much the same task as reinforcement learning in general. What is typical of RRL is the use of a relational (first-order) representation to represent states, actions and (learned) policies. Relational learning methods, originating from the field of inductive logic programming (Lavrac & Dzeroski, 1994), are used as generalization engines.

4.1 Task definition

While the task definition for reinforcement learning (as specified earlier in this paper) applies to RRL, a few details are worth noting. States and actions are represented relationally. Background knowledge and declarative bias need to be specified for the relational generalization engines.

The possible states would not be listed explicitly as input to the RRL algorithm (as they might be for ordinary reinforcement learning). A relational language for specifying states would rather be defined (in the blocks world, this language would comprise the predicates $on(A, B)$ and $clear(C)$). Actions would also be specified in a relational language ($move(A, B)$ in the blocks world) and not all actions would be applicable in all states; in fact the number of possible actions may vary considerably across different states.

Background knowledge generally valid about the domain (states in S) can be specified in RRL. This includes predicates that can derive new facts about a given state. In the blocks world, a predicate $above(A, B)$ may define that a block A is above another block B. Declarative bias for learning relational representations of policies can also be given. Together with the background knowledge, this specifies the language in which policies are represented. In the blocks world, e.g., we do not allow policies to refer to the exact identity of blocks ($A = a$, $B = b$, etc.).

4.2 The RRL algorithm

The RRL algorithm is obtained by combining the classical Q-learning algorithm (Table 1) and a relational regression tree algorithm (Blockeel et al., 1998). Instead of an explicit lookup table for the Q-function, an implicit representation of this function is learned in the form of a logical regression tree, called a Q-tree. After a Q-tree is learned, a classification tree is learned that classifies actions as optimal or non-optimal. This tree, called a P-tree, is usually much more succinct than the Q-tree, since it does not need to distinguish among different levels of non-optimality.

Table 3: The RRL algorithm for relational reinforcement learning.

Initialize $\hat{Q}_0$ to assign 0 to all (s, a) pairs
Initialize Examples to the empty set.
e := 0
while true
  generate an episode that consists of states s_0 to s_i
    and actions a_0 to a_{i-1} through the use of
    a standard Q-learning algorithm,
    using the current hypothesis for $\hat{Q}_e$
  for j=i-1 to 0 **do**
    generate example $x = (s_j, a_j, \hat{q}_j)$,
      where $\hat{q}_j := r_j + \gamma max_{a'} \hat{Q}_e(s_{j+1}, a')$
    if an example $(s_j, a_j, \hat{q}_{old})$ exists in Examples,
      replace it with x,
      else add x to Examples
  update $\hat{Q}_e$ by applying TILDE to Examples,
    i.e., $\hat{Q}_{e+1}$ = TILDE(Examples)
  for j=i-1 to 0 **do**
    for all actions a_k possible in state s_j **do**
      if state action pair (s_j, a_k) is optimal
        according to $\hat{Q}_{e+1}$
      then generate example (s_j, a_k, c) where $c = 1$
      else generate example (s_j, a_k, c) where $c = 0$
  update $\hat{P}_e$: apply TILDE to the examples (s_j, a_k, c)
    to produce $\hat{P}_{e+1}$
  e := e + 1

The RRL algorithm is given in Table 3. In its initial implementation (Dzeroski et al., 1998), RRL keeps a table of state/action pairs with their current Q-values. This table is used to create a generalization in the form of a relational regression tree (Q-tree). The Q-tree is then the policy used to select actions to take by the agent. The reason the table is kept is the nonincrementality of the relational regression algorithm used.

In complex worlds, where states can have a variable number of objects, an exact Q-tree representation of the optimal policy can be very large and also depend on the number of objects in the state. For example, in the blocks world, a state can have a varying number of blocks: the number of possible values for the Q-function (and the complexity of the Q-tree) would depend on this number. Choosing the optimal action, however, can sometimes be very simple: in the unstacking task, we simply have to pickup a block that is on top of another block and put it on the floor. This was our motivation for learning a P-tree by generating examples from the Q-tree.

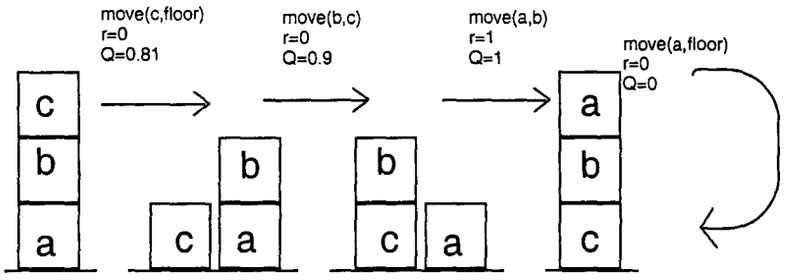


Figure 4: A blocks-world episode for relational Q-learning.

Table 4: Examples for TILDE generated from the blocks-world Q-learning episode in Figure 4.

Example 1	Example 2	Example 3	Example 4
qvalue(0.81).	qvalue(0.9).	qvalue(1.0).	qvalue(0.0).
move(c,floor).	move(b,c).	move(a,b).	move(a,floor).
goal(on(a,b)).	goal(on(a,b)).	goal(on(a,b)).	goal(on(a,b)).
clear(c).	clear(b).	clear(a).	clear(a).
on(c,b).	clear(c).	clear(b).	on(a,b).
on(b,a).	on(b,a).	on(b,c).	on(b,c).
on(a,floor).	on(a,floor).	on(a,floor).	on(c,floor).
	on(c,floor).	on(c,floor).	

4.3 An example

To illustrate how the RRL algorithm works, we use an example from the blocks world. The task here is to stack block a on block b, i.e., to achieve $on(a, b)$. An example episode is shown in Figure 4. As for the tabular version of Q-learning, updates of the Q-function are generated for all state/action pairs encountered during the episode. These are also listed in the figure.

The examples generated for TILDE from this episode are given in Table 4. Note that a Q-value of zero is assigned to any state/action pair where the state is terminal (the last state in the episode), as no further reward can be expected. From these examples, the Q-tree in Figure 5 is learned.

The tree correctly predicts zero Q-value if the goal is already achieved and a Q-value of one for any action, given that block A is clear. This is obviously overly optimistic, but does capture the fact that A needs to be clear in order to stack it onto B. Note that the goal $on(A, B)$ explicitly appears in the Q-tree.

If we use the Q-trees to generate examples for learning the optimality of actions, we obtain the P-tree in Figure 6. Note that the P-tree represents a policy much closer to the optimal one. If we want to achieve $on(A, B)$, it is optimal to move a block that is above A. Also, the action $move(A, B)$ is optimal whenever it is possible to take it.

4.4 Incremental RRL/TG trees

The RRL algorithm as described in the previous section has a number of problems. It needs to keep track of an ever increasing number of examples, needs to replace old Q-values with new ones if a state-action pair is encountered again, and builds trees from scratch after each episode. The G-tree algorithm (also mentioned earlier) does not have these problems, but only works for propositional representations. (Driessens et al., 2001) upgrade G-tree to work for relational representations yielding the TG-tree algorithm. At the top level, the TG-tree algorithm is the same as the G-tree algorithm. It differs in the fact that TG can use relational tests to split on; these are the same type of tests that TILDE can use. Using TG instead of TILDE within RRL yields the RRL-TG algorithm.

Table 5: The G-RRL algorithm: This is the RRL-TG algorithm with integrated guidance (k example traces).

Initialise $\hat{Q}_0$ to assign 0 to all $(state, action)$ pairs
for $(i = 0; i < k; i++)$ {
  transform $trace_i$ into $(state, action, qvalue)$ triplets
  process generated triplets with TG algorithm
    transforming $\hat{Q}_i$ into $\hat{Q}_i + 1$
  }
run normal RRL-TG starting with $\hat{Q}_k$ as the
  initial Q-function hypothesis

4.5 Integrating experimentation and guidance in RRL

Since RRL typically deals with huge state spaces, sparse rewards are indeed a serious problem. To relieve this problem, (Driessens & Dzeroski, 2002) follow the example of (Smart & Kaelbling, 2000) and integrate experimentation and guidance in RRL. In G-RRL (guided RRL), traces of human behavior or traces generated by following some reasonable policy (that could be learned previously) are provided at the beginning and are followed by ordinary RRL. The algorithm is given in Table 5.

```
root :  goal_on(A,B) , numberofblocks(C) , action_move(D,E)
on(A,B) ?
+--yes:  [0]
+--no:  clear(A) ?
        +--yes:  [1]
        +--no:  clear(E) ?
                +--yes:  [0.9]
                +--no:  [0.81]
```

Figure 5: A relational regression tree (Q-tree) generated by TILDE from the examples in Table 4.

```
root :  goal_on(A,B) , numberofblocks(C) , action_move(D,E)
above(D,A) ?
+--yes:  optimal
+--no:  action_move(A,B) ?
        +--yes:  optimal
        +--no:  nonoptimal
```

Figure 6: A P-tree for the three blocks world generated from the episode in Figure 4.

5 Experiments

Here we summarize the results of experiments with RRL. RRL was extensively evaluated experimentally on the blocks world by (Dzeroski et al., 2001). We first summarize these results. We then proceed with an overview of the most recent experiments with RRL, which involve the use of guidance in addition to pure reinforcement learning (Driessens & Dzeroski, 2002), i.e., the use of the G-RRL algorithm. These experiments involve the three domains described earlier in this paper: the blocks world, the Digger game and the Tetris game.

5.1 Blocks world experiments with RRL

(Dzeroski et al., 2001) conduct experiments in the blocks world with 3, 4, and 5 blocks, considering the tasks of stacking, unstacking and $on(a, b)$ mentioned earlier. They consider both settings with a fixed number of blocks (either 3, 4 or 5) or a varying number of blocks (first learn with 3 blocks, use this to bootstrap learning with 4 blocks, and similarly learn with 5 blocks afterwards). In addition to the state and action information, the RRL algorithm was supplied with the number of blocks, the number of stacks and the following background predicates: $equal/2$, $above/2$, $height/2$ and $difference/3$ (an ordinary subtraction of two numerical values).

The experiments show that RRL is effective for different goals: it was successfully used for stacking and unstacking, and after some representational engineering also for $on(a, b)$. Policies learned for $on(a, b)$ can be used for solving $on(A, B)$ for any A and B. Both can learn optimal policies for state spaces with a fixed number of blocks (both with Q- and P-trees), but this becomes more difficult when the number of blocks increases. An explanation for this is that the sparse rewards problem becomes more and more severe as the number of possible states skyrockets with an increasing number of blocks.

Even when learning from experience with a fixed number of blocks, RRL can learn policies that are optimal for state spaces with a varying number of blocks. Q-functions optimal for state spaces with a fixed number of blocks are not optimal for state spaces with a varying number of blocks. But we can learn optimal P-functions from the Q-functions. These P-functions often are optimal for state spaces with a varying number of blocks as well. RRL can also learn from experience in which the number of blocks is varied. Starting with a small number of blocks and gradually increasing this number allows for a bootstrapping process, where optimal policies are learned faster.

If the Q-tree learned doesn't work, then the P-tree won't work either. But once a Q-tree is learned that does the job right (even for states with a fixed number of blocks), one is better off using the P-tree learned from it. The latter usually generalizes nicely to larger numbers of blocks than seen during training.

5.2 Experiments with G-RRL

The experiments with G-RRL involve the three domains described earlier: the 10-blocks world, the Digger game and the Tetris game. Only Q-trees were built.

The blocks world

In the blocks world, the three tasks mentioned earlier (stacking, unstacking and $on(a, b)$) were addressed. Traces of the respective optimal policies were provided at the beginning of learning, followed by an application of the RRL-TG algorithm.

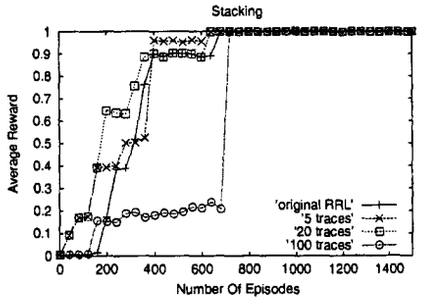


Figure 7: The learning curves of RRL and G-RRL for the stacking task.

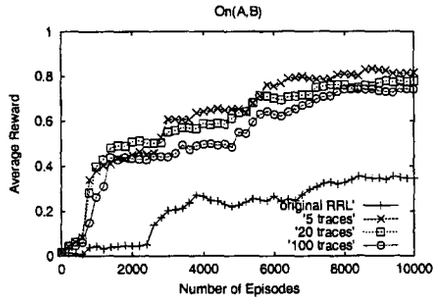


Figure 8: The learning curves of RRL and G-RRL for the on(a,b) task.

In summary, a moderate number of optimal traces helps the learning process converge faster and/or to a higher level of performance (average reward).

The learning curves for the stacking and $on(a, b)$ problems are given in Figures 7 and 8. G-RRL is supplied with 5, 20, and 100 of optimal traces. Providing guidance clearly helps in the $on(a, b)$ case, but less improvement is achieved when more traces are provided.

For stacking, better performance is achieved when providing 5 or 20 traces. However, providing 100 traces actually causes worse performance as compared to the original RRL algorithm. The experiment takes longer to converge and during the presentation of the 100 traces to G-RRL no learning takes place. The problem is that we supply the system with optimal actions only and it overgeneralizes, failing to distinguish between optimal and nonoptimal actions.

The Digger game

In the Digger Game, in addition to the state and action representation mentioned earlier, predicates such as $emerald/2$, $nearestEmerald/2$, $monster/2$, $visibleMonster/2$, $distanceTo/2$, $getDirection/2$, $lineOfFire/1$, etc., were provided as background knowledge for the construction of the Q-tree. Since it is hard to write an optimal policy, we used a policy generated in earlier work (Driessens & Blockeel, 2001) by RRL (which already performed quite well).

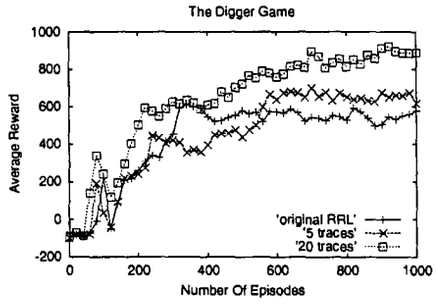


Figure 9: Learning curves for RRL and G-RRL for the Digger game.

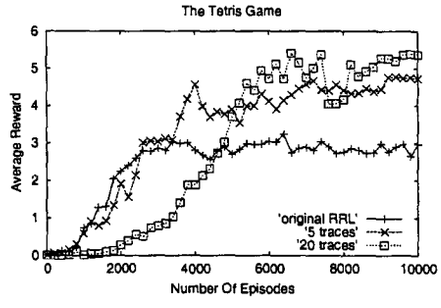


Figure 10: Learning curves for RRL and G-RRL for the Tetris game.

Figure 9 shows the average reward obtained by the learned strategies over 640 digger test-games divided over the 8 different Digger levels. It shows that G-RRL is indeed able to improve on the policy learned by RRL. Although the speed of convergence isn't improved, G-RRL reaches a significantly higher level of overall performance.

The Tetris game

For the Tetris game, RRL could use the following predicates (among others): $blockwidth/2$, $blockheight/2$, $rowSmaller/2$, $topBlock/2$, $holeDepth/2$, $holeCovered/1$, $fits/2$, $increasesHeight/2$, $fillsRow/2$ and $fillsDouble/2$. Like with the Digger Game, it is very hard (if not impossible) to generate an optimal or even "reasonable" strategy for the Tetris game. This time, we opted to supply G-RRL with traces of nonoptimal playing behavior from a human player.

The results for learning Tetris with RRL and G-RRL are below our expectations. We believe that this is due to the fact that the future reward in Tetris is very hard to predict, especially by a regression technique that needs to discretize these rewards like the TG algorithm. However, even with these disappointing results, the added guidance in the beginning of the learning experiment still has its effects on the overall performance. Figure 10 shows the learning curves for RRL and G-RRL supplied with 5 or 20 manually generated traces. The data points are the average number of deleted lines per game, calculated over 500 played test games.

6 Discussion

Relational reinforcement learning (RRL) is a powerful learning approach that allows us to address problems that have been out of reach of other reinforcement learning approaches. The relational representation of states, actions, and policies allows for the representation of objects and relations among them. Background knowledge that is generally valid in the domain at hand can also be provided to the generalization engine(s) used within RRL and adds further power to the approach.

We expect RRL to be helpful to agents that are situated in complex environments which include many objects (and possibly other agents) and where the relations among objects, between the agent and objects and among agents are of interest. The power of the representation formalism used would allow for different levels of awareness of other agents, i.e., social awareness (Kazakov & Kudenko, 2001). Knowledge about the existence and behavior of other agents can be either provided as background knowledge or learned.

There are many open issues and much work remains to be done on RRL. One of the sorest points at the moment is the generalization engine: it turns out that G-trees and TG-trees try to represent all policies followed by the agent during its lifetime and can thus be both large and ineffective. Developing better incremental and relational generalization engines is thus a priority. Finding better ways to integrate exploration and guidance also holds much promise for RRL. Finally, we are seeking to apply RRL to difficult, interesting and practically relevant problems.

Bibliographic notes

This paper summarizes research on relational reinforcement learning that has previously been published elsewhere. Relational reinforcement learning (RRL) was introduced by (Dzeroski et al., 1998) and further extended and experimentally evaluated on the blocks world by (Dzeroski et al., 2001). (Driessens et al., 2001) replaced the non-incremental generalization engine in RRL the with TG-tree algorithm, a relational version of the G-algorithm, yielding the RRL-TG algorithm.

(Driessens & Blockeel, 2001) applied RRL to the Digger game problem. (Driessens & Dzeroski, 2002) extended RRL-TG to take into account guidance from existing reasonable policies, either human generated or learned. They applied G-RRL to the Digger and Tetris games.

Acknowledgements

The author would like to thank Hendrik Blockeel, Kurt Driessenes and Luc De Raedt for the exciting and productive cooperation on the topic of relational reinforcement learning. Special thanks to Kurt Driessens for some of the figures and results included in this paper.

References

Bertsekas, D.P., & Tsitsiklis, J.N. (1996). *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific.

Blockeel, H., De Raedt, L., & Ramon, J. (1998). Top-down induction of clustering trees. In *Proc. 15th International Conference on Machine Learning*, pages 55–63. San Francisco: Morgan Kaufmann.

Chapman, D., & Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisions. In *Proc. 12th International Joint Conference on Artificial Intelligence*, pages 726–731. San Mateo, CA: Morgan Kaufmann.

Driessens, K., & Blockeel, H. (2001). Learning Digger using hierarchical reinforcement learning for concurrent goals. In *Proc. 5th European Workshop on Reinforcement Learning*, pages 11-12. Utrecht, The Netherlands: CKI Utrecht University.

Driessens, K., & Džeroski, S. (2002) Integrating experimentation and guidance in relational reinforcement learning. Submitted for publication.

Driessens, K., Ramon, J., & Blockeel, H. (2001). Speeding up relational reinforcement learning through the use of an incremental first order decision tree algorithm. In *Proc. 12th European Conference on Machine Learning*, pages 97–108. Berlin: Springer.

Džeroski, S., De Raedt, L., & Blockeel, H. (1998). Relational reinforcement learning. In *Proc. 15th International Conference on Machine Learning*, pages 136–143. San Francisco, CA: Morgan Kaufmann.

Džeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning, 43*, 7–52.

Kazakov, D., & Kudenko, D. (2001). Machine learning and inductive logic programming for multi-agent systems. In Luck, M., Marik, V., Stepankova, O., and Trappl, R., editors, *Multi-Agent Systems and Applications*, pages 246–270. Berlin: Springer.

Lavrač, N. and Džeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications.*, New York: Ellis Horwood.

Smart, W. D., & Kaelbling, L. P. (2000). Practical reinforcement learning in continuous spaces. *In Proc. 17th International Conference on Machine Learning*, pages 903–910. San Francisco, CA: Morgan Kaufmann.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Adaptive Agents to Heterogeneous Platforms, New Protocols & Evolving Organizations

Laurent Magnin* Viet Thang Pham* Arnaud Dury* Nicolas Besson*
Houari Sahraoui**

CRIM, 550 Sherbrooke Street West, Suite 100
Montréal, Québec, Canada H3A 1B9
{firstname.lastname}@crim.ca
Tel. +1 (514) 840 1234 - Fax. +1 (514) 840 1244

* Université de Montréal, Québec, Canada
sahraouh@iro.umontreal.ca

Abstract

Adaptive agents provide a natural and clean way to deal with heterogeneous, distributed and unpredictably evolving environments, such as Internet. In this paper, we present a model of agents that are able to adapt themself to their environment by several ways. Using a middleware approach, our agents, called *Guest*, can run, communicate and move between different multiagent platforms, which are *a priori* incompatible. *Guest* agents can also change dynamically their capabilities, based on the concept of plug-ins. To build multiagent applications, *Guest* agents are able to organize themselves into centralized or distributed hierarchies. These agents can even change automatically their organizational models to adapt to their evolving environment, using metamodeling technique.

1 Introduction

Multiagent systems are probably one of the most suitable approaches to build applications in open, heterogeneous, evolving and distributed environments, such as the Internet (7) (8). Being autonomous, agents running in such complex environments for a long period of time need to be self-adaptive to different platforms and new protocols. To bring this important caracteristic to agents, instead of a top-down approach where the main focus is on functionalities or behavior adaptation, we follow a bottom-up approach: it is useless to provide learning capabilities to agents if they don't have first basic tools to deal with such environments.

This paper will present some of the means we are using to achieve this vision. First, we propose a model of agents that are able to run, communicate and move between different multiagent platforms. This model is based on a middleware between such agents and platforms (section 2). We then describe briefly the implementation of this model on a kind of agents called *Guest*, which can be used the same way regardless of the kind of servers they run on (section 3). Next, we introduce a framework called *plug-ins* that allows agents to dynamically add, remove or upgrade their capabilities, such as understanding of new communication language, reasoning algorithm, etc. (section 4). Third, we propose two kinds of dynamic agent hierarchies, one is based on a middleware for the centralized context, while the other is designed using plug-ins for the distributed scenario (section 5). Lastly, we present a metamodel of agent control mechanisms to allow such agents to automatically adapt to their environments using the tools we early described (section 6).

2 Adaptation to heterogeneous multiagent platforms

An agent could only run on its own platform. For example, an Aglets agent can only use an Aglets based server, not a Grasshopper based one. The opposite is also true. Considering limited multiagent applications, this is not a real problem. However, future agents running in Internet-based open applications will have to be able to adapt themselves to heterogeneous servers provided by different partners.

Three feasible solutions to this interoperability problem can be envisioned: the use of standards (3) (11), the use of converters between platforms (15) and the creation of a generic interface (middleware approach). As (2), we choose the last one: we implement our generic agents (9) by using interfaces (see figure 1), more precisely by providing an intermediate layer called *Guest* between our *Guest* agents and the targeted agent platforms (all Java

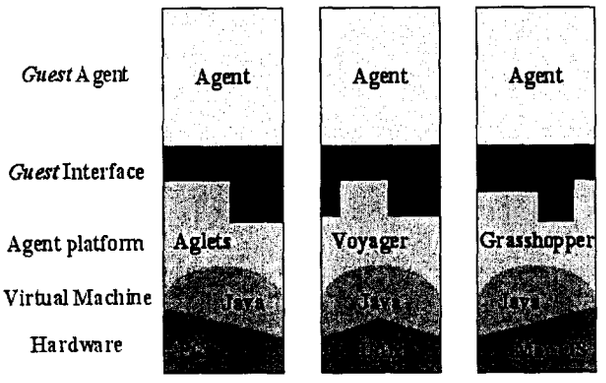


Figure 1: *Guest* interface

based). This layer is a two-sided entity: on one side, the *Guest* API that is visible to an application programmer, on the other side a platform-dependent layer, which we provide. In that way, our agent will be able to run and eventually talk to native agents on these different agent platforms while maintening the same functionality thanks to interfaces (one per platform). That does not imply that all of the platforms need to integrate these *Guest* interfaces: it is only when a *Guest* agent reaches an agent server that the *Guest* Java classes need to be downloaded from a specific server by the Java Virtual Machine (JVM) without modifying the server behavior. This method does not impose a new standard, or more precisely parties who want to use the genericity of the *Guest* API do not require that *Guest* becomes a standard. This point is essential to the success of our approach.

A *Guest* agent is made up of two facets: one is specific to the platform expected to carry the agent, the other one is independent of any platform. Moreover, in order for this *Guest* agent to move from one platform to another, it should be able to change dynamically its specific facet while maintaining its internal status. It is generally impossible to get access to the source code of agents associated with the targeted platforms. As a result, it is impossible to implement these generic agents by a modification of their code which is offered by the platforms. We must resort to interfacing with the public methods proposed by these agents. Therefore, our solution consists in designing a *Guest* agent modeled as the sum of two interconnected agents. The first one (so-called native agent) inherits from the agent class of the targeted agent platform. The second one (so-called generic agent with the purpose of implementing the agent function) inherits from the generic agent class *Guest*. It is the only part of the agent that will move between servers when the agent migrates [1]. Consequently, our agents are simultaneously perceived by platforms as being native and by their originators as being platform-independant *Guest* agents. Our model requires slightly more resources than a native model: memory consumption and CPU overhead are limited, but are far from being doubled.

A critical constraint has to be mentioned: all the selected platforms require using their own Java Virtual Machines. When a composite *Guest* agent migrates, the generic part will be handed over to the targeted JVM right after the native part is created on it. Upon completion, the composite agent will be removed from its originating platform, thus ending the migration process. One of the conditions for the agent migration is to ensure that the *Guest* agent is serializable.

As mentioned above, our *Guest* agents must have *Guest* interfaces adapted to specific multiagent platforms if they are intended to be operational. There are already *Guest* interfaces on those Java based platforms:

- ASDK, Aglets Software Development Kit (1), originally developed by the IBM Tokyo Research Laboratory.
- CorbaHost, our own Java server implementation on top of Corba (Orbix by IONA(4)) [2].
- Grasshopper (5), commercial product of German firm IKV++.
- Jade (6), an open source platform which is Fipa compliant [3].
- Voyager (12), commercial product of ObjectSpace.

3 Generic Guest Agents

Our goal is not only to have agents that can run on different platforms but also not to care about the platform they run on. In other words, the agent programmer will work with the uniform interface we provide, but will not have to deal with specific code for the different targeted platforms. For example, the agent programmer can use the same addressing schema to represent addresses of different servers, regardless of their platforms. We achieve this by providing a complete and operational set of agent features through the methods provided by the GuestAgent class. Of course, the API of such methods is the same regardless which platform the agent is running on.

3.1 How to deal with specific features of the platforms ?

By providing always the same interface to our *Guest* agents, do we restrict the features allowed to them to the most

[1] As is the case on many other agent platforms, the current state of the running agent will not been saved and restored when the agent migrates from one server to another. It is the duty of the agent programmer to solve this problem by using specific methods which are called just before and after the migration

[2] We developped this server to offer the bridge between our *Guest* agents and the world of CORBA.

[3] We specially choose this platform to provide Fipa communication capabilities to our *Guest* agents.

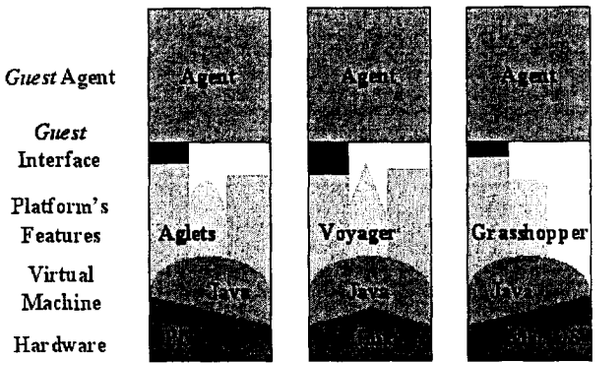


Figure 2: Equivalent features provided by native platforms

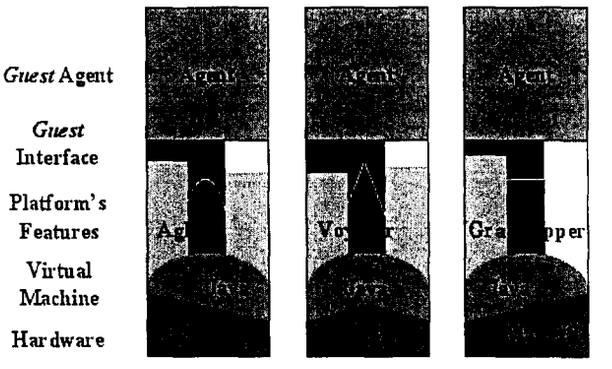


Figure 3: Different features provided by native platforms

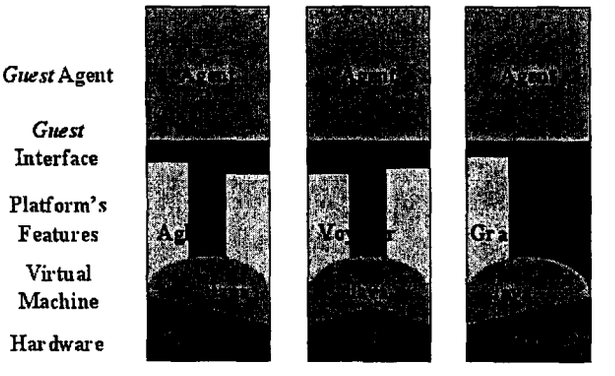


Figure 4: Lack of features provided by native platforms

common denominator of the different platforms they can run on? In other words, in order to be provided by *Guest* should a feature be available on all targeted platforms? If so, only limited functionalities would be given to the agents. Also, only platforms providing all of *Guest* current features would be able to be added to the targeted platforms.

Fortunately, this constraint does not hold. To explain how we deal with this problem, we have to divide the features in three sets. The first one contains features that are quite similar for all platforms. For example, Aglets, Grasshopper and Voyager provide the ability to send a message to an agent by calling a proxy.function(args) like method, so, the *Guest* interface can implement this functionality simply by calling the native method of the targeted platform. In figure 2, we show that the *Guest* code (the black square on the first column) is merely a simple function call. In the second case, a feature is different on the targeted platforms. For example, the naming services are different for Aglets, Grasshopper and Voyager. To solve this problem it is possible to implement this feature independently, inside the interface. In figure 3, we show that *Guest* implements the same functionality on the three differents platforms (the functionality being the black area on the second column of fig. 3). That way, *Guest* provides a uniform type of addresses for agents running on these three platforms.

Last case, a feature can be unavailable on some platforms. For example, Voyager does not provide a function that returns all of the agents running on a specific server. Such a function can be implemented from scratch for this platform. At the same time, this function can also be based on a call to the native functions provided by Aglets and Grasshopper. In figure 4, we show in the third column that the *Guest* implementation (the black area of the figure) varies from platform to platform, from a simple function call (on aglets and voyager) to a full implementation (on grasshopper).

In conclusion, our middleware approach allows us to provide a uniform set of functionalities to universal agents despite the fact that the targeted platforms could have various features and goals.

3.2 Guest Agent Model

3.2.1 Agent's life cycle

No matter on what server a *Guest* agent executes, the agent can be in one of the following states:

- STATUS_CREATION : agent is built, but has not been initialized and can not communicate with other agents.
- STATUS_EXECUTION : agent finishes initialization, can communicate with other agents and can be fully used.
- STATUS_MIGRATION: a mobile agent is in this state when it is on the way to its destination. Such a moving agent can not directly receive and process messages, but all the messages sent to the agent during this period are buffered and later forwarded to it.
- STATUS_SAVE: when an agent stops temporarily its work and saves itself on disk, it is changed to

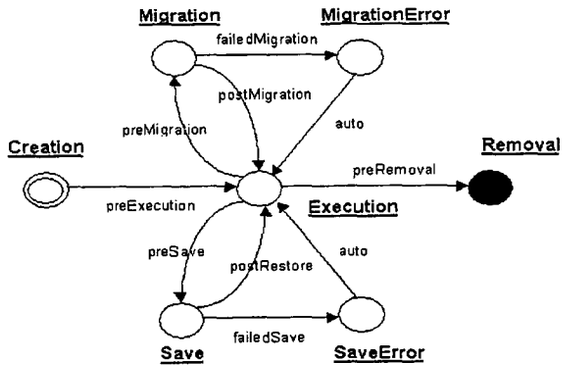


Figure 5: The agent life cycle

this state. An agent at this state can not receive messages and does not do anything. However, such an agent can be woke up by itself or by another agent to continue its task.

- STATUS_REMOVAL: agent enters this state when it prepares to die. An agent in this state can not be moved or saved.

The other two states: MIGRATION_ERROR and SAVE_ERROR are temporal states and are used internally by GuestAgent.

This life cycle model is the most common part of the different life cycles of the four targeted platforms. Moreover, it is sufficient to express different states and to support all the basically necessary functionalities of a universal agent. For example, states of a JADE agent, which strictly follows the Agent Platform Life Cycle in FIPA specification, can be easily mapped into states of our model: AP_INITIATED → STATUS_CREATION, AP_ACTIVE → STATUS_EXECUTE, AP_DELETED → STATUS_REMOVAL, AP_TRANSIT → STATUS_MIGRATION. AP_SUSPENDED and AP_WAIT should be represented as a substate of the STATUS_EXECUTE, because an agent in these states is still alive and the actions of SUSPEND and WAIT can only be activated by an agent on itself.

3.2.2 Agent URL

In the *Guest* platform, the address of an agent is uniformly represented by a GuestURL. A GuestURL has the syntax of a common URL: **platform_protocol://host_name — IP_address:port[/optional]**

- ***platform_protocol*** represents the protocol supporting the specific native platform. For example, *Guest* platform supports Grasshopper, Voyager, Aglets, CorbaHost[4] and JADE by offering the following protocols: GH, VY, AG, CH and JD, respectively. Based on this signature, agent can correctly activate the appropriate native part to accomplish its task ;

[4]CorbaHost is a small agents server on top of Corba we created.

- ***host_name*** is the domain name of the computer on which the agent executes. It can be replaced by the IP address of this computer ;
- ***port*** is an integer indicating the port on which the native server is listening ;
- ***optional*** is used to represent the platform-dependant extension of the GuestURL. For example, the Grasshopper agent needs to include Place and Region in its address, this extension is therefore necessary.

This form has proved to cover all the representative forms of agent addresses of the four targeted platforms. For example, an Aglets server uses the following string to represent its address: "atp://halida.crim.ca:4434". This address can be easily transformed into the GuestURL as "AG:// halida.crim.ca:4434" and vice versa.

3.2.3 Agent communication

Any *Guest* agent can send and receive *Guest* messages through an uniform API. As the communication between Guest agents running on the same platform use the native layers provided by the platform, the communication between heterogenous platforms is based on Java RMI. This mechanism enables the delivery of *Guest* messages between heterogeneous platforms, but does not allow the processing of native messages exchanged between non-*Guest* agents. Such capability is nevertheless possible using direct access to the native agent, which is also supported by the uniform API. However, that code is platform-specific and may not be executed on all platforms. *Guest* platform supports three communication modes between agents: synchronous, asynchronous and future reply. While the first two modes use the communication service of the native platform, the last one is completly implemented in *Guest* based on the synchronous communication mode.

4 Adaptive agents based on plug-ins

In an open, evolving multiagent world, new algorithms and services will be put into use during the life-cycle of an agent or system. For example, a new compression algorithm may be used to compress messages, or a new kind of cryptographic signature may be released. To adapt to that kind of environments, our agents need to be able to apply "on the fly" these new algorithmes and services, without having to restart any part of the system. More precisely, we want to be able to design "new capabilities" in an agent-independent way, and then allow our *Guest* agents to use them *on demand*.

Guest agents are currently able to operate on heterogeneous platforms, thanks to the interface layer on top of native agents. This interface layer encompass creation and destruction of the native agent, message sending and receiving, migration, etc and is therefore sufficient in terms of agents programming. But it is fixed at the runtime and

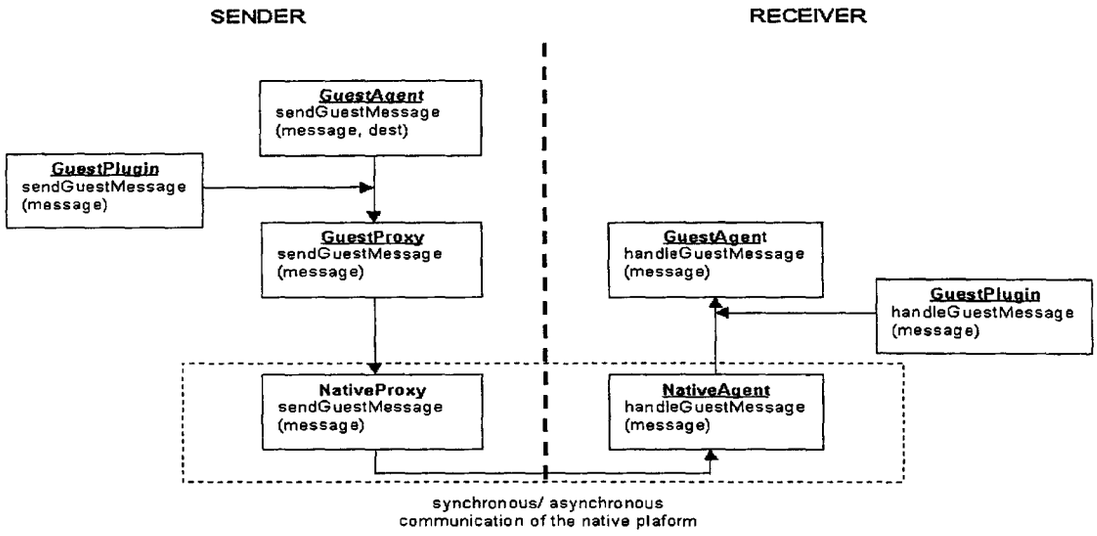


Figure 6: Two base communication modes

not suitable for dynamically adding new capabilities to an existing agent: any modification implied to this interface requires the restart of all the agent servers on which *Guest* agent are running. Therefore, we need a more convenient way to solve this problem. Our solution is to embed in the kernel of the *Guest* agent a framework called *plug-in*. A plug-in is a component (compiled code of one or some objects) which can be associated with an agent and immediately offer some new services. This framework has several clear advantages:

- Enable Guest agent to modify dynamically its capabilities and therefore adapt to the evolving environment. For example, one agent can "learn" to compress/decompress data by loading a compression plug-in. When it no longer needs these functions, it can simply "forget" it by unloading the plug-in. The agent can even "upgrade" its compression technique by changing the current compression plug-in for a more sophisticated plug-in;
- Enable reuse and sharing of services: a compression plug-in can be developped and deployed by a third party. It can then be used as an *Off-The-Shelf-Component*.

A plug-in can perform any of the following three types of actions:

- Observe the change of the associated agent state and possibly prevent this change in some cases (migration or deactivation of the agent);
- Observe the communication of the associated agent (sending and receiving a message) and possibly intercept an incoming/outgoing message;
- Offer a library of new services to the agent. These services can be used by indicating the name of the service and all the necessary parameters, or by first obtaining a plug-in reference and then accessing to the services using normal function call on this reference.

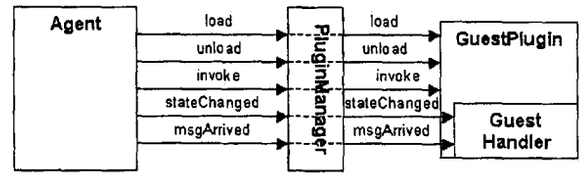


Figure 7: Plug-in framework

The relationship between a plug-in and an *Guest* agent is clearly defined in the base class GuestPlugin. This class contains necessary methods for the load, unload and discovery of a plug-in. Every plug-in must extend this class in order to cooperate correctly with the agent and offer its services to the agent user. All the new services (high-level functionality) of a plug-in are exposed to the agent user through its reference and can be easily accessed by simple function calls. Moreover, these services can be normally used outside the agent via the associated agent's proxy, without having to modify this one. All the actions of observing/intercepting agent state/communication events are low-level functionality of a plug-in and separately handled by a "secret part" of the plug-in, called *handler*. A handler makes part of a plug-in and is totally hidden from outside the plug-in in order to prevent unattended and unauthorized access. Every handler must extend the base class GuestHandler and implements some of the prede-

fined interfaces to declare what kind of events it is interested in. For example, a handler which implements the interface ISynGuestMessageHandler will be automatically called by the associated agent each time a synchronous communication event is fired.

Moreover, the mecanism of event handling is not a mere chaining, but is in fact more sophisticated. Upon loading, a plug-in exports to the agent its *activation function*. This function is a filter, applied on incoming messages and events, deciding on which ones the plug-in offers its services. For example, a dedicated decompression plug-in would ask for activation only when compressed messages are received. It would then ask for activation for each potentially encapsulated compressed message. Any given plug-in can be used any number of times during the processing of a message or an event. This allows us to provide new capabilities (for instance compression and cryptography), even if we don't know in advance the proper layering of the message by the other agents (do they first sign, and then compress? Or do they compress, and then sign?). This is a very important capability in the kind of open environment we developed *Guest* for.

Our design ensures the maximum dynamicity for the modularization by allowing a plug-in to be associated with an agent at any time during the agent's life and removed whenever the user no longer needs it. The deployment of these plug-ins is also flexible: one plug-in doesn't have to be bundled in the packages but can be downloaded from an Internet site.

Using the plug-in framework, we already developed a graphical interface that allows a special user to visually observe the Guest agents on the network (using different native platforms), see figure 8.

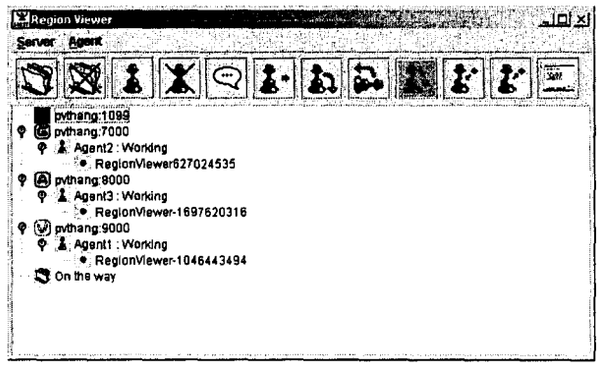


Figure 8: Guest Graphical interface

This was done by creating plug-in which intercepts the changes of agent's state during the migrations and informs the GUI to visually reflect these changes on the screen. Consequently, an agent can be observed only by associating with it this plug-in even if this agent was not initially supposed to offer this service. By the same way, it will be possible to allow a *Guest* agent to be compatible with standards of agent interactions, such as KQML(19), Fipa (3) or any new standard that can be defined in the future.

In the next section, we will show how to use the plug-in framework to intergrate the distributed hierarchical agent model in *Guest*.

5 Adaptive hierarchical multiagent systems

Not only agents need to be adaptive, but also organizations of agents need to be. Working on different kind of such organizations, we will focus on this paper on a specific one: agents' hierarchies[5], which can be dynamically modified.

We developed two different ways of building such hierarchies that are described in the following sections. The first one uses our middleware approach by implementing at the agent level the platform interface: each agent can now be seen as an agent server, which is able to receive and run other agents. The second one uses a specific plug-in we developed that can dynamically modify the routing of messages between agents to offer the functional appearance of a hierarchy.

5.1 Agents as servers

5.1.1 Principles

In the context of the *Guest* platform, an agent server is a container which is able not only to provide resources (such as CPU time) and communication service to other agents, but also to monitor and control the life cycle of these agents. We have designed *Guest* agents to be able to act as servers (parent agent), i.e. they can accept and execute other agents (children agents). These internal agents in turn can also be servers for other agents and so on recursively. Finally we have a group of agents organized into a hierarchy. In this hierarchy, only the root agent needs to be connected to the native platform, through the *Guest* interface. The other agents are just connected to their corresponding parent agents through the parent/child interface, which is mostly the same as the *Guest* interface.

This hierarchical model simplifies the design of a multiagent system, particularly for those whose processing can be divided into hierarchical tasks: the native platform sees a hierarchy as a unique agent, and the outside world may be given a unique entry-point (the root agent) for the hierarchy. The parent agent has total control on its children: migration, destruction, etc. and the root agent is thus able to control all the agents in the hierarchy. For example, when an application has to migrate from one server to another (because of harware failure, for example) it only needs to ask the root agent to move, then the

[5] A hierarchy in our platform is a direct acyclic graph of binary parent/child relations.

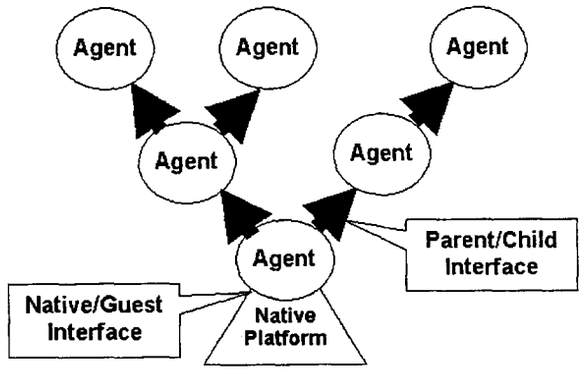


Figure 9: Agents as servers

whole hierarchy will move together. The organization is maintained and the application resume its work normally right after it arrives at the new destination. The communication between parent and child agents is local and very efficient. The consumed resources are considerably reduced because only one native agent is required for the whole hierarchy. The hierarchical model is thus one of the most natural ways to "agentify" a whole multiagent systems, except that it requires that all the agents in the hierarchy reside on the same server.

5.1.2 Implementation

This hierarchical model is implemented in the kernel of the *Guest* platform. To represent the structure of the hierarchy, the concept of GuestURL is extended to encapsulate not only the address of the server of the agent but also the path from the root of the hierarchy to this agent. That way, integration of an agent into a hierarchy is done by a migration where the URL of the target is another agent instead of a real server. The life cycle of the children is monitored and intercepted by a mechanism similar to the one used by the plug-in. All the messages sent to a child will be firstly received by its parent and then be forwarded (according to the parent policy) to it. We added a new interface into the GuestAgent to allow it to manage children' communication and their execution threads, this interface is similar to the native agent interface but has some new specific functions for the hierarchy : add, remove, etc. The child can be dynamically connected to its parent through this interface and has the impression that it is fostered by a server. The communication between the parent and its children is implemented as direct function calls, thus greatly increasing the speed of exchanging messages (compared to the usage of the "send a message" primitive).

To overcome the limit that all the agents must be on the same server, we propose in the next section another hierarchical model, using rerouting technique.

5.2 Dynamic rerouting

5.2.1 Principles

Rerouting messages between agents is the other method we use to model the creation of dynamic hierarchies between multiple cooperating agents. This mechanism allows a group of agents to organize themselves into a more efficient multi-agent structure by dynamically rerouting messages received by the structure. The main idea behind this concept of rerouting is: in order to function as a hierarchy, a group of agents needs to have a leader (root of the direct acyclic graph describing the hierarchy) that receive and dispatch all the messages for the group (this process is recursive, as is the case with the previous hierarchy model). If an agent in a hierarchy, that is not the root of this hierarchy, receives a message then it reroutes it to the root of its hierarchy. We are in fact replicating the way the previous model of hierarchy (see 5.1) works, but in this new model, agents don't need to be physically on the same server. The rerouting of messages will duplicate the way the previous hierarchy works, without its constraints of physical locality. This model is not without its flaw: the number of messages exchanged will increase, compared to the previous model of hierarchy (see 5.1).

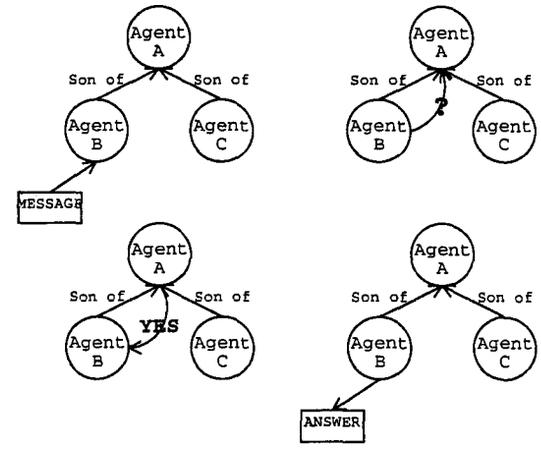


Figure 10: Building a hierarchy using dynamic rerouting

5.2.2 Implementation

This dynamic rerouting is implemented by a specific plug-in, the Hierarchical plug-in. This plug-in offers the following services:

- ask a potential father to register the agent (create a new Father-Child link)
- ask the father to unregister oneself (Child ask to destroy the link its Father)
- obtains the list of all the childs of the agent
- find whether or not a given agent is a children

- find whether or not a given agent is a grand-children
- allow each agent to express its acceptance policy for accepting a new child
- allow each agent to express its acceptance policy for letting a child leaving the hierarchy
- allow the Guest graphical interface to display the hierarchy as a tree

Group creation is implemented by a two-way handshake between the agent willing to join the group, and his desired leader in this group. The agent sends a request to the desired leader, and upon acceptance reroutes all of his incoming messages to this leader for approval. This mechanism, being used in a recursive way, leads to the creation of a hierarchy between agents.
Group destruction follows the same principle: each agent willing to leave the group generates a request to its direct father. This request can be forwarded through child to father to grandfather etc. up to the root of the hierarchy. The acceptance policy is based on the acceptance of each father, from the agent up to the root (that is, each father can express a veto on the decision). Upon entering a hierarchy, an agent looses its ability to process directly any incoming message. It will from now on have to ask its direct father for the authorization to process the incoming message (see figure 10). This ensures a uniform processing of the request by a unique agent (the leader), even if the outside agents are not yet aware of the new organization, and erroneously send their request to an agent inside the group. We now introduce a possible use of this model, considering the following context: how to add load-balancing capability to an application ?

5.2.3 Load-balancing example

A capacity that is usually not straightforward to implement for a system, but is easily implemented using this kind of hierarchy, is the capacity of load-balancing for an application. Load-balancing is the process by which a computation requiring lots of ressources (memory or CPU time usually) is distributed among a network of computers, according to the load of each computer, and the computation itself. Using Guest agents and hierarchies, a simple way of implementing load-balacing is feasible: an agent is created to realize a piece of the computation (for instance, an agent that is able to render a pixel, in a raytracing application). This agent loads the hierarchy plugin we describe in 5.2.1. This agent then connects to a GuestRegion[6] to discover the other servers. As this agent receives pixel-rendering requests from an outside (potentially non-agent oriented) application, it creates clones of itself, and these clones register themselves as children to the first agent. Each clone is able to move to a server with a lighter load than its own. The first created agent will be the only agent known to the outside world, and will process requests and dispatch them among pixel-rendering agents. These pixel-rendering agents won't respond to any other agent, and would automatically forward to their father any incoming message. In this context Guest provides a way to keep track of all pixel-rendering agents, to allow them to choose the server with the mininum load while keeping their link with the dispatcher agent and to enforce the rule that rendering agents will respond only via their dispatcher father.

[6] A GuestRegion is a specific service of Guest that allows agents to track existing agents and servers.

5.3 Designing a hierarchy using both approaches

Hierarchy is a useful way of modeling certain execution strategies in multi-agent systems. We described in the precedent section the use of hierarchies to add load-balancing capability to an application. This use of the concept of hierarchy (and its current implementation in Guest) can be refined even further. The distributed hierarchy we described in figure 10 is inefficient in terms of number of messages exchanged: for each message received by an agent inside the hierarchy, another one is generated to forward the incoming message to the root of the hierarchy. We describe now the inefficiency problem associated with distributed hierarchy when the number of agents is inferior to the number of servers, and a way to optimize the efficiency of a distributed hierarchy by combining it with a centralized hierarchy each time two or more agents are on the same server.

5.3.1 The problem: inefficiency of the distributed hierarchy

Suppose we have three servers, and ten agents in a hierarchy, distributed among these servers. We cannot use the highly efficient (in terms of messages exchanged) centralized hierarchy,because we have to use several servers, and the distributed hierarchy that we have to use is less efficient in terms of number of messages exchanged (due to the re-routing process). The solution we now describe is simply to combine both.

5.3.2 Combining both hierarchies

We introduce a new execution strategy, that allows us to automatically minimize the number of messages exchanged inside a hierarchy. The strategy is simply to adapt the kind of link between two agents according to the servers they are on. The hierarchic link between two agents on the same server will be of the first kind (see 5.1), whereas the link between two agents on different servers will be of the second kind (see 5.2). When an agent in a hierarchy migrates, the type of link with its own parent is changed. Using both hierarchies, we are now able to minimize the number of messages exchanged at the strict minimum, because we no longer have to reroute message between

agents on the same server. We discuss in section 6 the use of meta-modelling to allow for automatic adaption of the hierachy.

6 Self-adaptive agents by metamodeling

Until now, we presented "base bricks", which allow to modify, even in the course of execution, the control mechanisms and the functionalities associated with a *Guest* agent. However, the question of the layout of these bricks still remains unanswered. How to prevent, for example, two plug-ins, which are incompatible, to be used at the same time within the same agent? How to ensure that the use of one particular plug-in automatically leads to the installation of another plug-in, essential to the first one?

The first approach to solve these issues (approach which we could called ascendance) would consist in describing all these constraints in the form of rules. This approach could prohibit a certain number of configuration or add/remove some plug-ins associated with the agent, when certain particular conditions or actions are met.

However this solution seems too limited to be able to deal with complex and especially nonforeseeable configurations, such as the case with the Internet. This is why we prefer the descendant approach, based on the description of the agent in the form of the abstract models which can be instantiated *in fine*. Furthermore, such models allow a greater generalization to take place.

In order to more easily be able to manipulate these models, we should represent them in a uniform fashion. In other words, we should model them in a form which would become the *de facto* metamodel (18) of the agents. Our work is to research the mechanisms allowing to model the transformation from one model to another, and these changes, of course, can be defined by the same (static) metamodel. This metamodel should allow not only to describe all the possible transformations, but also to enable the correct transformation of one agent from a state corresponding to a given model to another state described by another model.

The principal of our model is the horizontal separation between the "control" part of the agent and its "function" part, based on the research in (21) (20) and validated in the development process of the platform Guest. The "control" part is application-independant and presents in all or most agents. For example, the communication or the capacity of migration of an agent, or even the agent model itself, can be classified as "control" part. Meanwhile, the capacity of learning or reasoning of an agent belongs to the "function" part, because it is application-dependant or at least domain-dependant. If our model can support automatic adaptation to the "control" part of agent, which is possible because this part is application-independant, the task of making an adaptive agent is much more easier and can be semi-automatically done. For example, we already have two models of hierarchical agent, one is most appropriate in the centralized scenario, while the other is preferable in the distributed context. The rest of agents (perception, deliberation, action) is unchanged. If the "control" part of agents is adaptive, agents can therefore transparently "switch" from one hierarchical model to the other model when the scenario changes from centralized to distributed one or vice versa.

The figure 11 presents our model of agent, which distingushes between "control" and "function" part, along with the concept of plug-in.

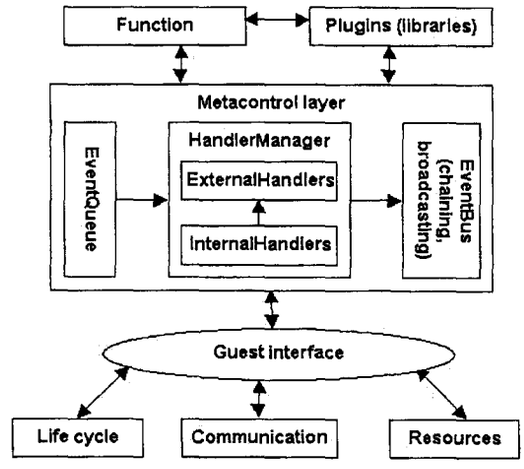


Figure 11: Metamodel of agent

In this model, the Metacontrol layer plays a crucial role, that is to connect different pieces of agent control, agent function and agent services (plug-ins) in the correct manner. Its architecture is inspired by the Java InfoBus technology (22), the semantic of the connection between different parts is captured in the types of events and in the handler's type. We will demonstrate how this model works in the case of two types of hierarchical agent model.

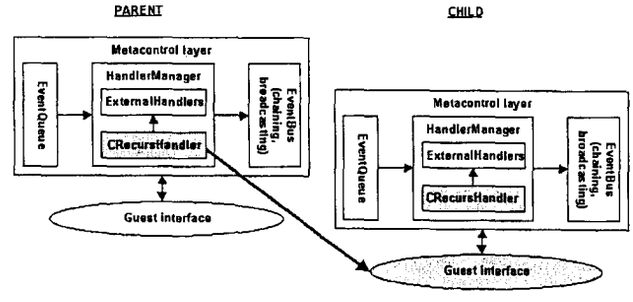


Figure 12: Centralized hierarchical agent

The model of centralized hierarchical agent is represented in the figure 12. In this model, the CRecursHandler is an internal handler, the bold line indicates that this handler is viewed by the child agent as an agent server.

The parent and child agent are in the same JVM and all the communications between parent and child are direct function call, thus very efficient. The CRecursHandler will control all the events related to the creation, destruction, migration and communication of the child agent.

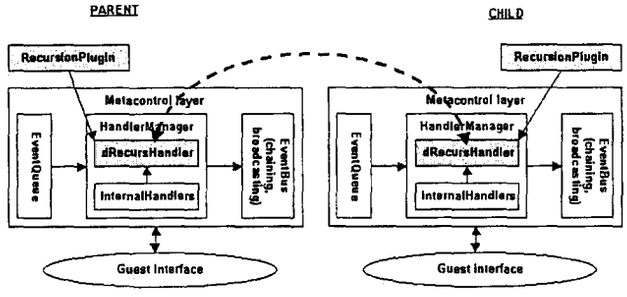


Figure 13: Distributed hierarchical agent

In the model of the distributed hierarchical agent, agent needs to use an external handler, DRecursHandler, which can be downloaded from a Web site and activated immediately after it is initialized. All the communications from/to the child agent are intercepted by this handler and rerouted to the parent agent to be authorized. The dotted link indicates that the rerouting is done through network communication.

As a consequence, the dynamic transformation from one model to another model is just simple as the activation/desactivation of the corresponding handler. We are now working on the next step, that is to offer agents the faculty to automatically adapt themselves to their environment by using these transformations.

For example, when an agent, or even a branch, of the centralized hierarchy recognizes that the server on which it resides is overloaded, it wants to move to another less busy server, while assuring the integrity of the hierarchical organization by changing to the distributed hierarchy.

In order to do that, agent needs to use a third handler, which is external and handles events of type "CPU overloaded". Whenever an agent receives a signal from the server that it is overloaded, this handler will be activated and realize the transformation of the agent's hierarchical model.

7 Conclusion

Today's multiagent applications must deal with new constraints of open and complex environments, of which the Internet is the most typical example. The development of these applications requires that agents become generic and adaptive to their evolving environment, i.e. neither linked to specific execution platforms or to a unique standard, nor fixed to a limited set of models and services.

This is why, in this paper, we have presented a new model of uniform agents, called *Guest*, using the middleware approach. Based on a generic interface, *Guest* agent applications are able to execute the same way without worrying about the incompability between different agent platforms on which agents are running. To be adaptive, *Guest* agents can dynamically update their capabilities at runtime by adding or removing *on demand* plug-ins dedicated to specific tasks like the authorization of migration or the handle of secured communication. Our *Guest* agents also supports two organizational models: centralized and distributed hierarchy. Finally, we provide a metamodel which allows agents to automatically change their control mechanisms, such as organizational model, to adapt to their environment.

Guest is already a full-fledged prototype with support for interoperability between Aglets, Corba, Jade, Grasshopper and Voyager platforms. The plug-ins framework is fully functional, both of the hierarchical models are implemented and currently tested. We are working on the implementation and validation of the presented metamodel. Applications of the *Guest* platform in the real world currently include a completed work in the industrial building management context.

8 Acknowledgements

This work is done at the CRIM laboratory and mainly founded by it. We have also to acknowledge the efforts of Nekrouf Ziani for HostCorba implementation and of Ivan Guentchev for his experiments on Jade, which were, as other students who worked on this project, supported by Laurent Magnin's personal Research Grant from the Natural Sciences and Engineering Research Council of Canada.

References

[1] Aglets Workbench, http://aglets.sourceforge.net/

[2] Control of Agent Based Systems, http://coabs.globalinfotek.com/

[3] Fipa, Foundation for Intelligent Physical Agents, http://www.fipa.org/

[4] Orbix by IONA, http://www.iona.com/ Concurrent Objects: Briot-Gasser Interview" http://www.lis.uiuc.edu/ gasser/AgentsAndObjects-07.html

[5] Grasshopper, Grasshopper by IKV++, http://www.grasshopper.de/

[6] The Java Agent DEvelopment Framework, http://jade.cselt.it/

[7] Joshi, A. and M. P. Singh (1999). Multiagent Systems on the Net. Communications of the ACM 42 (March 1999) pp. 39-49. Communications of the

ACM 42 (March 1999) pp. 79-80. R. H. Guttman, et al., Agents That Buy and Sell, Communications of the ACM 42 (March 1999) pp. 81-91. First International Workshop on Mobile Agents for Telecommunication Applications (MATA 99), Ottawa, Canada, October 6-8, 1999

[8] Magnin L., "Internet, environnement complexe pour agents situés, Proceedings of the Intelligence artificielle situe conference, Paris, October 25-26, 1999, pp.213-221.

[9] Magnin L. and Alikacem E.H., Guest: Multiplatform Generic Agents, Proceedings of the First International Workshop on Mobile Agents for Telecommunication Applications (MATA 99), Ottawa, Canada, October 6-8, 1999, pp. 507-514.

[10] Magnin L., et al., "Our Guest agents are welcome to your agent platforms", The 17th ACM Symposium on Applied Computing, Special Track on Agents, Interactions, Mobility, and Systems (AIMS), Madrid, Spain, March 10 - 14, 2002. l'organisation dans les SMA. Application la conception des Syst?mes Coopratifs Distribus et Ouverts, Ph.D. thesis, University Paris VI, Paris, 1998. Marketplace, Proceedings of Autonomous Agents, Seattle, ACM, May 1999.

[11] OMG, MASIF: Mobile Agent System Interoperability Facility, http://www.fokus.gmd.de/research/cc/ima/masif/index.html

[12] Voyager, Voyager by ObjectSpace, http://www.objectspace.com/voyager/

[13] Wong, D., N. Paciorek, et al., Java-based Mobile Agents, Communications of the ACM 42 (March 1999) pp. 92-102.

[14] Wooldridge, M., Intelligent Agents, Multiagent Systems (ed. by G. Weiss, Cambridge, London, The MIT Press, 1999) pp. 27-78.

[15] Tjung D., Tsukamoto M. and Nishio S., A converter Approach for Mobile Agent System Integration: A Case of Aglet to Voyager, Proceedings of the First International Workshop on Mobile Agents for Telecommunication Applications (MATA 99), Ottawa, Canada, October 6-8, 1999, pp. 179-195.

[16] William Harrison and Harold Ossher, Subject-oriented programming: a critic of pure objects, in Proceedings of OOPSLA 93, Washington D.C., Sept. 26-Oct 1, 1993, pp. 411-428.

[17] Gregory Kiczales, John Lamping, Cristina Lopez, Aspect-Oriented Programming, in Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. Springer-Verlag LNCS 1241. June 1997.

[18] Revault N., Sahraoui H.A., Blain G., Perrot, J.-F. "A Metamodeling Technique: The MétaGen System". Tools Europe proceedings, (1995).

[19] KQML, http://www.cs.umbc.edu/kqml

[20] Min-Young Yoo, Une approche componentielle pour la modélisation d'agent coopératifs et leur validation. Thèse de doctorat de l'université Paris VI, 2000.

[21] Jacques Ferber, Les systèmes multi-agents, InterEditions, 1996.

[22] http://java.sun.com/products/javabeans/infobus.

Dynamic Adaptation of Replication Strategies for Reliable Agents

Jean-Pierre Briot; Zahia Guessoum; Sébastien Charpentier;
Samir Aknine; Olivier Marin; Pierre Sens
LIP6, Université de Paris 6
BP 169, 4 place Jussieu, F-75252 Paris Cedex 5
Jean-Pierre.Briot@lip6.fr; Zahia.Guessoum@lip6.fr; charpentier@mangoosta.net;
Samir.Aknine@lip6.fr; Olivier.Marin@lip6.fr; Pierre.Sens@lip6.fr

Abstract

To make large-scale multi-agent systems reliable, we propose an adaptive application of replication strategies. Critical agents are replicated to avoid failures. As criticality of agents may evolve during the course of computation and problem solving, we need to dynamically and automatically adapt the number of replicas of agents, in order to maximize their reliability and availability based on available resources. We are studying an approach and mechanisms for evaluating the criticality of a given agent (based on application-level semantic information, e.g. messages intention, and also system-level statistical information, e.g., communication load) and for deciding what strategy to apply (e.g., active replication, passive) and how to parameterize it (e.g., number of replicas).
In this paper, we first present the replication mechanism and the framework named DarX that we developed to replicate agents. We then describe a new model to evaluate dynamically the criticality of agents. Then we describe the implementation of this model with the DarX fault-tolerant framework.

1 Introduction

A multi-agent system is a set of autonomous and interactive entities called agents (Avouris and Gasser, 1992). Recent real-life applications (e.g., intensive care monitoring, air traffic control and process control) are often distributed and must run continuously without any interruption. As a distributed system, however, multi-agent systems are exposed to possibility of failure of their hardware and/or software components. The failure of one component can often evolve into the failure of the whole system. To make these large-scale multi-agent systems reliable, an obvious solution is the introduction of redundancy: duplication (replication) of the critical components.

Replication mechanisms have been successfully applied for various distributed applications (Guerraoui and Schiper, 1997), e.g. data-bases. But in most cases, replication is decided by the programmer and applied statically, before the application starts. This works fine because the criticality of components (e.g., main servers) may be well identified and remains stable during the application run.

Opposite to that, in the case of dynamic and adaptive multi-agent applications, the criticality of agents may evolve dynamically during the course of computation. Moreover, the available resources are often limited. Thus, simultaneous replication of all the components of a large-scale system is not feasible. The idea is then to *automatically* and *dynamically* apply replication mechanisms *where* (to which agents) and *when* it is most needed. In this paper we will describe our approach to this objective and the realized tool to build easily reliable multi-agent systems.

This paper is organized as follows. Section 2 presents fault tolerance concepts and replication principles. Section 3 introduces a new approach of dynamic control of replication. Section 4 presents the DarX framework that we developed to replicate agents. This framework introduces novel features for dynamic control of replication. Section 5 describes our approach to compute agent criticality in order to guide replication. Section 6 describes the implementation of this solution and our preliminary experiments.

2 Fault-Tolerance

2.1 First and Simple Example

We consider the example of a distributed multi-agent system that helps at scheduling meetings. Each user has a personal assistant agent which manages his calendar. This agent interacts with:

- the user to receive his meeting requests and associated information (a title, a description, possible dates, participants, priority, etc.) ,
- the other agents of the system to schedule a meeting.

If the assistant agent of one important participant (initiator or prime participant) in a meeting fails (e.g., his ma-

chine crashes), this may disorganize the whole process. As the application is very dynamic - new meeting negotiations start and complete dynamically and simultaneously - decision for replication should be done automatically and dynamically.

2.2 Type of Faults

To achieve fault-tolerance, several distributed systems replicate their critical components. In this section, we define a classification of failures in multi-agent systems.

In computing systems, faults can for example occur in different sections of source code, and can result in a wide range of consequences. In distributed systems, another kind of faults can be considered: the communication failures.

A fault classification scheme is often used to categorize faults that have the same characteristics. The defined categories can then be used to collect statistics about faults and devise methods for fault prevention and detection. Thus, several classifications of failures have been proposed.

A. Fedoruk and R. Deters propose a classification of failures in multi-agent systems (Fedoruk and Deters, 2002). They group failures in five categories:

- Program bugs,
- Unforeseen states,
- Processor faults,
- Communication fault,
- Emerging unwanted behavior.

However, it is not easy to define the unwanted states of an agent because he is often adaptive. Moreover, a multi-agent system has no global control. So, the emerging unwanted behavior can be detected by an external observer, but it cannot be easily determined automatically by the system itself.

In our proposed solution, we consider, two categories of failures:

- Processor faults,
- Communication fault.

We think that our approach could also be applied to other categories of failures.

2.3 Techniques of Replication

This section, first, summarizes the principles of replication. It then points out the limits of current replication techniques and replication tools.

2.3.1 Principles of Replication

Replication of data and/or computation is an effective way to achieve fault tolerance in distributed systems. A replicated software component is defined as a software component that possesses a representation on two or more hosts (Guerraoui et al., 1989). There are two main types of replication protocols:

- active replication, in which all replicas process concurrently all input messages,
- passive replication, in which only one of the replicas processes all input messages and periodically transmits its current state to the other replicas in order to maintain consistency.

Active replication strategies provide fast recovery but lead to a high overhead. If the degree of replication is n, the n replicas are activated simultaneously to produce one result.

Passive replication minimizes processor utilization by activating redundant replicas only in case of failures. That is: if the active replica is found to be faulty, a new replica is elected among the set of passive ones and the execution is restarted from the last saved state. This technique requires less CPU resources than the active approach but it needs a checkpoint management which remains expensive in processing time and space.

The active replication provides a fast recovery delay. This kind of technique is dedicated to applications with real-time constraints which require short recovery delays. The passive replication scheme has a low overhead under failure free execution but does not provide short recovery delays. The choice of the most suitable strategy is directly dependent on the environment context, especially the failure rate, and the application requirements in terms of recovery delay and overhead. Active approaches should be chosen either when the failure rate becomes too high or the application design specifies hard time constraints. Otherwise, passive approaches are preferable.

2.3.2 Limits of Current Replication Techniques

Many toolkits (e.g., (Guerraoui et al., 1989) and (van Renesse et al., 1996)) include replication facilities to build reliable applications. However, most of them are not quite flexible enough to implement adaptive replication mechanism.

Therefore we designed a specific and novel framework for replication, named DarX (see details in section 4), which allows dynamic replication and dynamic adaptation of the replication policy (e.g., passive to active, changing the number of replicas).

3 Towards Dynamic Replication and Adaptive Control

Several solutions have been proposed to replicate distributed systems. These solutions are often used by the designer to replicate the system components before run time. The number of replicas and the replication strategy are explicitly and statically defined by the designer before run time. However, these solutions are not suitable to multi-agent systems. The solution we propose is mainly characterized by dynamic replication and adaptive control.

3.1 Dynamic Replication

Several replication strategies (mainly, active and passive) can be used to replicate Agents. As explained in Section 2.3.1, each strategy has its pros and cons, the tradeoff being recovery speed versus overhead. Thus, the choice of the most suitable strategy relies on the environment context.

In most multi-agent applications, the environment context is very dynamic. So, the choice of the replication strategy of each component, which relies on a part of this environment, must be determined dynamically and adapted to the environment changes.

Moreover, a multi-agent system component which can be very critical at a moment can loose its critically later. If we consider the replication cost which is very high, the number of replicas of these components must be therefore dynamically updated.

Thus, the solution we propose allows to dynamically adapt the number of replicas and the replication strategy. This solution is provided by the framework DARX (see section 4).

3.2 Adaptive Control

DarX provides the needed adaptive mechanisms to replicate agents and to modify the replication strategy. Meanwhile, we cannot always replicate all the agents of the system because the available resources are usually limited. In the given example (section 2.1), we can consider more than 100 assistant agents and resources that do not allow to duplicate more than 60 agents. The problem therefore is to determine the most critical agents and then the needed number of replicas of these agents.

We distinguish two cases:

- the agent's criticality is static,
- the agent's criticality is dynamic.

In the first case, multi-agent systems have static organization structures, static behaviors of agents, and a small number of agents. Critical agents can be therefore identified by the designer and can be replicated by the programmer before run time.

In the second case, multi-agent systems may have dynamic organization structures, dynamic behaviors of agents, and a large number of agents. So, the agents criticality cannot be determined before run time. The agent criticality can be therefore based on these dynamic organizational structures. The problem is how to determine dynamically these structures to evaluate the agent criticality? Thus, we propose a new approach for observing the domain agents and evaluating dynamically their criticality. This approach is based on two kinds of information: semantic-level information and system-level information.

4 Darx

DarX is a framework to design reliable distributed applications which include a set of distributed communicating entities (agents). Each agent can be dynamically replicated an unlimited number of times and with different replication strategies (passive and active).

4.1 Darx Architecture

DarX includes group membership management to dynamically add or remove replicas. It also provides atomic and ordered multi-cast for the replication groups' internal communication. For portability and compatibility issues, DarX is implemented in Java.

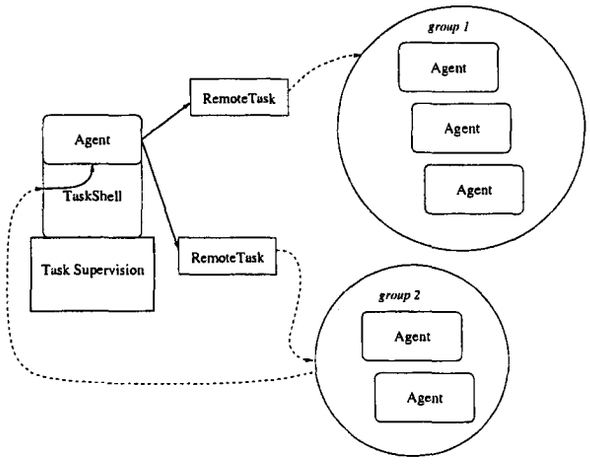


Figure 1: DarX application architecture

A replication group is an opaque entity underlying every application agent. The number of replicas and the internal strategy of a specific agent are totally hidden to the other application agents. Each replication group has exactly one leader which communicates with the other agents. The leader also checks the liveness of each replica and is responsible for reliable broadcasting. In case of failure of a leader, a new one is automatically elected among the set of remaining replicas.

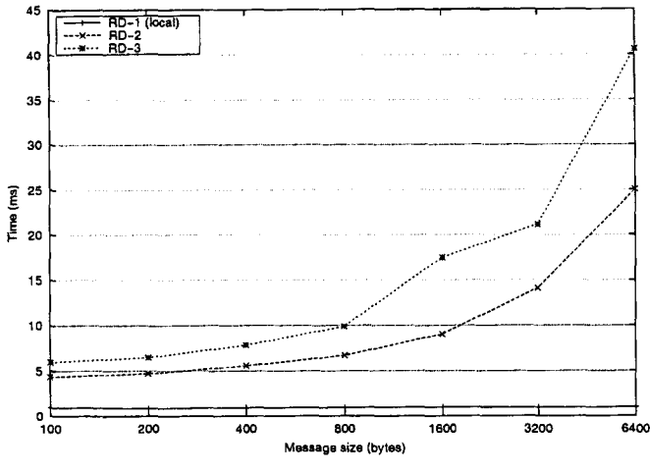


Figure 2: Communication cost as a function of the replication degree

DarX provides global naming. Each replicated agent has a global name which is independent of the current location of its replicas. The underlying system allows to handle the agent's execution and communication. Each agent is itself wrapped into a TaskShell (Figure 1), which acts as a replication group manager and is responsible for delivering received messages to all the members of the replication group, thus preserving the transparency for the supported application. Input messages are intercepted by the TaskShell, enabling message caching. Hence all messages get to be processed in the same order within a replication group.

An agent can communicate with a remote agent, unregarding whether it is a single agent or a replication group, by using a local proxy implemented by the RemoteTask interface. Each RemoteTask references a distinct remote entity considered as the leader of its replication group. The reliability features are thus brought to agents by an instance of a DarX server (DarxServer) running on every location. Each DarxServer implements the required replication services, backed up by a common global naming/location service.

4.2 Measurements

Our first experiments and measurements of DarX are very promising. We evaluated several costs and made comparisons with other systems (see (Marin et al., 2001)).

In this paper, we just show the cost of sending a message to a replication group using the active replication strategy. Figure 2 presents three configurations with different replication degrees. In the RD-1 configuration, the task is local and not replicated. In the RD-2 (resp. RD-3) configuration, there is one (resp. two) replica(s); the leader being on the sending host and the other replica(s) residing on one (or two) distinct remote host(s).

5 Adaptive Control of Replication

We will now detail our approach for dynamically evaluating criticality of each agent in order to perform dynamic replication where and when best needed.

5.1 Hypothesis and principles

We want some automatic mechanism for generality reasons. But in order to be efficient, we also need some prior input from the designer of the application. This designer can choose among several approaches of replication: static and dynamic.

In the proposed dynamic approach, the agent criticality relies on two kinds of information:

- System-level information. It will be based on standard measurements (communication load, processing time...). We are currently evaluating their significance to measure the activity of an agent.
- Semantic-level information.

Several aspects may be considered (importance of agents, independence of agents, importance of messages...). We decided to use the concept of role, because it captures the importance of an agent in an organization, and its dependencies to other agents.

Note that our approach is generic and that it is not related to a specific interaction language or application domain. Also agents can be either reactive or cognitive. We just suppose that they communicate with some agent communication language such as ACL (FIPA, 1997) and KQML (Finin et al., 1994).

5.2 Example

The application designer will manually evaluate criticality of the roles, corresponding to their "importance" in the organization and in the computation.

In the example introduced in section 2.1, we are considering two roles: Initiator and Participant (Finin et al., 1994). Their respective weights will be set by the application designer to respectively 0.7 and 0.3 (see 1).

Table 1: Examples of roles and their weights

Roles	Weights
Initiator	0.7
Participant	0.3

5.3 Architecture

In order to track the dynamical adoption of roles by agents, we propose a role recognition method. Our approach is based on the observation of the agent execution and their

interactions to recognize the roles of each agent and to evaluate his processing activity. This is used to dynamically compute the criticality of an agent.

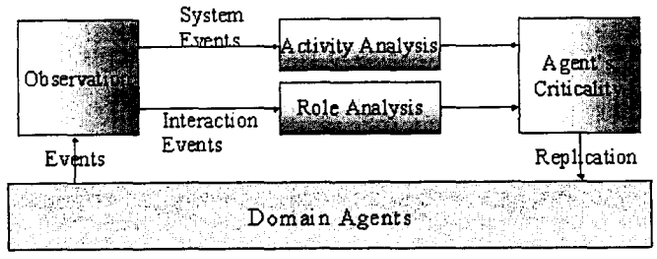


Figure 3: General architecture for replication control

In order to collect the data, we associate an observation module to each DarxServer on each machine (see section 4). This module will collect events (provided by DarxServer). A role analysis is then associated to each agent (leader of the group replica) of this machine, by considering his sent and received messages.

The basic architecture controlling the replication of agents is shown in Figure 3.

The next sections describe the role analysis and activity analysis methods that we propose.

5.4 Role Analysis

We consider two cases. In the first case, each agent displays explicitly his roles or interaction protocols. The roles of each agent are thus easily deduced from interaction events. In the second case, agents do not display their roles nor their interaction protocols. The agent roles are deduced from the interaction events by the role analysis module.

In this analysis, attention is focused on the precise ordering of interaction events (exchanged messages). The role module captures and represents the set of interaction events resulting from the domain agent interactions (sent and received messages).

We associate to each agent an entity that analyses the associated interaction events. This analysis determines the roles of the agent. Figure 4 illustrates the various steps of this analysis.

To represent the agent interactions, several methods have been proposed such as state machines and Petri nets (Fallah-Seghrouchni et al., 1999). For our application, state machines provide a well suitable representation. Each role interaction model is represented by an augmented transition network (ATN) (Woods, 1970). A transition represents an interaction event (sending or receiving a message). Figure 5 shows an example of ATN that represents the interaction model of the role Initiator described below.

A library of roles definition is used to recognize the active roles. To facilitate the initialization of this library,

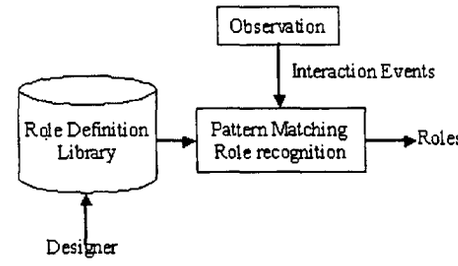


Figure 4: Roles recognition

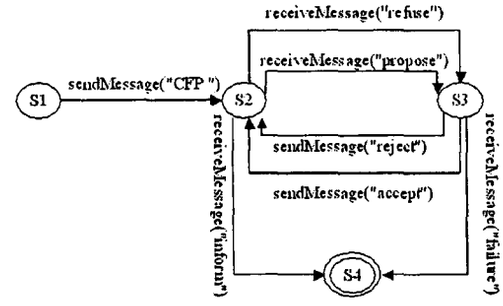


Figure 5: Example of ATN

we have introduced a role description language. Each role is represented by a set of interaction events. This language is based on a set of operators (similar to those proposed in (M. Wooldridge and Kinny, 1999), see Table 2), interaction events and variables.

Interaction events represent the exchanged messages. We distinguish two kinds of interaction events: ReceiveMessage and SendMessage. The attributes of the SendMessage and ReceiveMessage interaction events are similar to the attributes of ACL messages:

- SendMessage(Communicative act, sender, receiver, content, reply-with, ...).
- ReceiveMessage(Communicative act, sender, receiver, content, reply-with, ...).

Table 2: Operators

Operators	Interpretation
A.B	Separate two consecutive events
A|B	Or
A||B	Parallel events
(A)*	O time or more
(A)+	1 time or more
(A)n	n time or more
[A]	Facultative

In order to be able to filter various messages, we introduce the "wild card" character ?. For example, in the interaction event ReceiveMessage ("CFP", "X", "Y", ?), the content is unconstrained. So, this interaction event can match any other interaction event with the communication act CFP, the sender "X", the receiver "Y" and any contents.

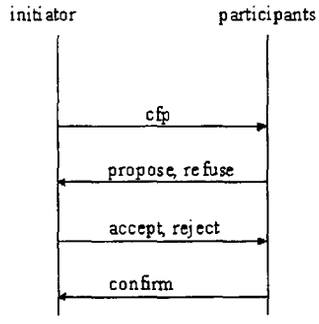


Figure 6: Contract net protocol

In the example of scheduling meetings, the assistant agents use the contract net protocol (FIPA, 1997) (see 6) to schedule a meeting. The interaction model of the initiator role is deduced from the contract net protocol. It is described in Table 3.

This description represents the different steps (sent and received messages) of the Initiator. It can be interpreted as follows (FIPA, 1997).

- A call for proposals message is sent to the participants from the initiator following the FIPA Contract Net protocol.
- The participants reply to the initiator with the proposed meeting times. The form of this message is either a proposal or a refusal.
- The initiator sends accept or reject messages to participants.
- The participants which agree to the proposed meeting inform the initiator that they have completed the request to schedule a meeting (confirm).

Table 3: Description of the role *Initiator*

(SendMessage("CFP", Agent,?,?,M1))+ .
((ReceiveMessageEvent("propose", ?, Agent,?,M2, M1)) |
(ReceiveMessageEvent("refuse", ?, Agent, ?,M2, M1)))+ .
((SendMessage("accept", Agent,?,?, M2, M1)) |
(SendMessage("reject", Agent,?,?, M2, M1)))+ .
(ReceiveMessageEvent("confirm", ?, Agent,?,M2))+ .

Note that in many cases, roles can be deduced before the end of the associated sequence of interaction events (final state of the associated ATN). In the scheduling meetings example, the role Initiator may be recognized as soon as the "CFP" message is received, as it is unique to this role.

5.5 Activity Analysis

In multi-agent systems, the internal activity of agents cannot be observed, because it is private. The observation is restricted to events. To evaluate the degree of the agent activity, we use system events that are collected at the system level. We are considering two kinds of events: CPU time and communication load. We are currently evaluating the significance of these measures as indicators of agent activity, to be useful to calculate agent criticality.

For an agent $Agent_i$ and a given time interval Δt, these events provide:

- The used time of CPU (cp_i),
- The communication load (cl_i).

cp_i and cl_i may be then used to measure the agent degree of activity aw_i as follows:

$$aw_i = (d_1 * cp_i/\Delta t + d_2 * cl_i/CL)/(d_1 + d_2) \quad (1)$$

where:

- CL is the global communication load,
- d_1 and d_2 are weights introduced by the user.

5.6 Agent Criticality

The analysis of events (system events and interaction events) provides two kinds of information: the roles and the degree of activity of each agent. This information is then processed by the agent's criticality module. The latter relies on a table T (an example is given in Table 1) that defines the weights of roles. This table is initialized by the application designer. Table 3 gives examples of roles and their weights.

The criticality of the agent $Agent_i$ which fulfills the roles r_{i1} to r_{im} is computed as follows:

$$w_i = (a_1 * \sum_{j=1,m} T[r_{ij}] + a_2 * aw_i)/(a1 + a2) \quad (2)$$

Where: a_1 and a_2 are the weights given to the two kinds of parameters (roles and degree of activity). They are introduced by the designer.

For each Agent A_i, its criticality w_i is used to compute the number of his replicas.

5.7 Replication

An agent is replicated according to:

- w_i: his criticality,
- W: the sum of the domain agents' criticality,
- rm: the minimum number of replicas which is introduced by the designer,
- Rm: the available resources which define the maximum number of possible simultaneous replicas.

The number of replicas nb_i of $Agent_i$ can be determined as follows:

$$nb_i = rounded(rm + w_i * Rm/W) \quad (3)$$

Table 4: Examples of agents, their weights and the associated number of replicas

Agents	Criticality per agent	Number of replicas per agent
Agent1, Agent2, Agent3, Agent 4	0,9	2
Agent5, Agent6, Agent7, Agent8, Agent9, Agent10, Agent11, Agent12, Agent13, Agent14	0.5	1
Agent15, Agent16, Agent17, Agent18, Agent19, Agent20, Agent21, Agent22, Agent23, Agent24	0.2	0

Table 4 gives an example of agents, their criticality and the associated replicas when Rm = 20 and rm= 0. Note that (rm=0) means that the agent is not replicated.

The numbers of replicas are then used by DarX to update the number of replicas of each agent.

6 Experiments

We made some preliminary experiments using the scenario of agents scheduling their meetings, as introduced in section 2.1.

Agents take randomly roles of Initiator, choose Participants for scheduling meetings or remain inactive (without any role). Several meetings are scheduled simultaneously. The number of critical agents (which can be either Initiator or Participant) is 60% of the number of agents.

As the distributed observation module implementation has not been completed yet, we have run these preliminary experiments on a single machine. In order to simulate the presence of faults, we implemented a failure simulator randomly stopping the thread of an agent (chosen randomly). The number of the introduced faults was set equal to the number of agents. We repeated several times the experiments with a variable number of resources (number of replicas that can be used).

From these first experiments, we found that the number of resources should be at least equal to the number of critical agents.

We are currently working on more experiments and measurements in order to better evaluate our adaptive control architecture and to compare it to other control methods (including random replication).

7 Related Work

Several approaches address the multi-faced problem of fault tolerance in multi-agent systems. These works can be classified in two main categories. A first approach focuses especially on the reliability of an agent within a multi-agent system. This approach handles the serious problems of communication, interaction and coordination of agents (and their replicas) with the other agents of the system. The second approach addresses the difficulties of making reliable an agent, particularly a mobile agent, which is more exposed to security problems (Pleisch and Schiper, 2001) (Silva and Popescu-Zeletin, 1998) (Johansen et al., 199) (Strasser et al., 1998). This second approach is beyond the scope of this paper.

Within the family of reactive multi-agent systems, some systems offer high redundancy. A good example is a system based on the metaphor of ant nests. Unfortunately:

- we cannot design any application in term of such reactive multi-agent systems. Basically we do not have yet a good methodology.
- we cannot apply such simple redundancy scheme onto more cognitive multi-agent systems as this would cause inconsistencies between copies of a single agent.

Some work (Decker et al., 1997) offers dynamic cloning of specific agents in multi-agent systems. But their motivation is different, the objective is to improve the availability of an agent if it is too congested. The agents considered seem to have only functional tasks (with no changing state) and fault-tolerance aspects are not considered.

S. Hagg introduces sentinels to protect the agents from some undesirable states (Hagg, 1997). Sentinels represent the control structure of their multi-agent system. They need to build models of each agent to perform functionalities and monitor communications in order to react to faults. Each sentinel is associated by the designer to one functionality of the multi-agent system. This sentinel handles the different agents which interact to achieve the functionality. The analysis of his believes on these agents enables the sentinel to detect a fault when it occurs. Adding

sentinels to multi-agent systems seems to be a good approach, however the sentinels themselves represent failure points for the multi-agent system.

(Kumar et al., 2000) present a fault tolerant multi-agent architecture that regroups agents and brokers. They address the problem of recovering the multi-agent system from only its broker failures.

(Fedoruk and Deters, 2002) propose to use proxies. This approach tries to make transparent the use of agent replication, i.e. enabling the replicas of an agent to act as a same entity regarding the other agents of the system which will not know that they are interacting with a group of replicas. The proxy manages the state of the replicas. To do so, all the external and internal communications of the group are redirected to the proxy. However this increases the workload of the proxy which is a quasi central entity. To make it reliable, they propose to build, for instance, a hierarchy of proxies for each group of replicas. They point out the specific problems of read/write consistency, resource locking also discussed in (Silva et al., 2000).

In distributed computing, many toolkits include replication facilities to build reliable application. However, many of products are not enough flexible to implement an adapted replication. MetaXa (M.Golm, 1998) implements in Java active and passive replication in a flexible way. Authors extended Java with a reactive metalevel architecture. Like in DarX, the replication is transparent. However, MetaXa relies on a modified Java interpreter. GARF (Guerraoui et al., 1989) realizes fault-tolerant Smalltalk machines using active replication. Similar to MetaXa, GARX uses a metalevel and provides different replication strategies. But, it does not provide adaptive mechanism to apply these strategies.

8 Conclusion

Large-scale multi-agent systems are often distributed and must run without any interruption. To make these systems reliable, we proposed a new approach to evaluate dynamically the criticality of agents. This approach is based on the concepts of roles and degree of activity. The agent criticality is then used to replicate agents in order to maximize their reliability and availability based on available resources.

To validate the proposed approach, we realized a fault-tolerant framework (Darx). The integration of DARX with a multi-agent platform, such as DIMA ieee99, provides a generic fault-tolerant multi-agent platform. In order to validate this fault-tolerant multi-agent platform, two small applications have been developed (meetings scheduling and crisis management system). They are intended at evaluating our model and architecture viability. They aim also at completing the model and adjusting the parameters. The obtained results are interesting and promising. However, more experiments with real-life applications are needed to validate the proposed approach.

References

N. A. Avouris and L. Gasser. *Distributed Artificial Intelligence: Theory and Praxis*, chapter Object-Oriented Concurrent Programming and Distributed Artificial Intelligence, pages 81–108. Kluwer Academic Publisher, 1992.

K. Decker, K. Sycara, and M. Williamson. Cloning for intelligent adaptive information agents. In *ATAL'97*, LNAI, pages 63–75. Springer Verlag, 1997.

A. El Fallah-Seghrouchni, S. Haddad, and H. Mazouzi. Protocol engineering for multiagent interactions. In *MAAMAW'99*, number 1647 in LNAI, pages 128–135. Springer Verlag, 1999.

A. Fedoruk and R. Deters. Improving fault-tolerance by replicating agents. In *AAMAS2002*, Boulogna, Italy, 2002.

T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an agent communication language. In *Third international conference on information and knowledge management*. ACM Press, November 1994.

FIPA. Specification. part 2, agent communication language, foundation for intelligent physical agents, geneva, switzerland. http://www.cselt.stet.it/ufv/leonardo/fipa/index.htm, 1997.

R. Guerraoui, B. Garbinato, and K. Mazouni. Lessons from designing and implementing garf. In *Proceedings Objects Oriented Parallel and Distributed Computatio*, volume LNCS 791, pages 238–256, Nottingham, 1989.

R. Guerraoui and A. Schiper. Software-based replication for fault tolerance. *IEEE Computer*, 30(4):68–74, April 1997.

S. Hagg. A sentinel approach to fault handling in multi-agent systems. In C. Zhang and D. Lukose, editors, *Multi-Agent Systems, Methodologies and Applications*, number 1286 in LNCS, pages 190–195. Springer Verlag, 1997.

D. Johansen, K. Marzullo, F. B. Schneider, K. Jacobsen, and D. Zagorodnov. Nap: Practical fault-tolerance for itinerant computations. In *19th IEEE International Conference on Distributed Computing Systems (ICDCS)*, Austin, Texas, 199.

S. Kumar, P. R. Cohen, and H. J. Levesque. The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams. In *The Fourth International Conference on Multi-Agent Systems ICMAS, Boston,USA*, 2000.

N. Jennings M. Wooldridge and D. Kinny. The methodology gaia for agent-oriented analysis and design. *AI*, 10(2):1–27, 1999.

O. Marin, P. Sens, J.-P. Briot, and Z. Guessoum. Towards adaptive fault-tolerance for distributedmulti-agent systems. In *ERSADS'2001*, pages 195–201, 2001.

M.Golm. Metaxa and the future of reflection. In *OOPSLA -Workshop on Reflective Programming in C++ and Java*, pages 238–256. Springer Verlag, 1998.

Stefan Pleisch and Andr Schiper. Fatomas - a fault-tolerant mobile agent system based on the agent-dependent approach. In *In Proceedings of the IEEE Int. Conf. on Dependable Systems and Networks (DSN'01)*, 2001.

F. De Assis Silva and R. Popescu-Zeletin. An approach for providing mobile agent fault tolerance. In S. N. Maheshwari, editor, *Second International Workshop on Mobile Agents*, number 1477 in LNCS, pages 14–25. Springer Verlag, 1998.

L. Silva, V. Batista, and J. Silva. Fault-tolerant execution of mobile agents. In *International Conference on Dependable Systems and Networks*, pages 135–143, 2000.

M. Strasser, K. Rothermel, and C. Maihofer. Providing reliable agents for electronic commerce. In W. Lamersdorf and M. Merz, editors, *Int. Conference on Trends in Distributed Systems for Electronic Commerce*, number 1402 in LNCS, pages 241–253. Springer Verlag, 1998.

R. van Renesse, K. Birman, and S. Maffeis. Horus: A flexible group communication syste. *CACM*, 39(4):76–83, 1996.

W. Woods. Transition network grammar for natural language analysis. *Communication of Association of Computing Machinery*, 10(13):591–606, 1970.

On Learning by Exchanging Advice

Luís Nunes
Eugénio Oliveira
LIACC-NIAD&R;
FEUP Av. Dr. Roberto Frias 4200-465, Porto, Portugal.
Luis.Nunes@iscte.pt; eco@fe.up.pt

Abstract

One of the main questions concerning learning in Multi-Agent Systems is: "(How) can agents benefit from mutual interaction during the learning process?". This paper describes the study of an interactive advice-exchange mechanism as a possible way to improve agents' learning performance. The advice-exchange technique, discussed here, uses supervised learning (backpropagation), where reinforcement is not directly coming from the environment but is based on advice given by peers with better performance score (higher confidence), to enhance the performance of a heterogeneous group of Learning Agents (LAs). The LAs are facing similar problems, in an environment where only reinforcement information is available. Each LA applies a different, well known, learning technique: Random Walk (hill-climbing), Simulated Annealing, Evolutionary Algorithms and Q-Learning. The problem used for evaluation is a simplified traffic-control simulation. In the following text the reader can find a description of the traffic simulation and Learning Agents (focused on the advice-exchange mechanism), a discussion of the first results obtained and suggested techniques to overcome the problems that have been observed. Initial results indicate that advice-exchange can improve learning speed, although "bad advice" and/or blind reliance can disturb the learning performance. The use of supervised learning to incorporate advice given from non-expert peers using different learning algorithms, in problems where no supervision information is available, is, to the best of the authors' knowledge, a new concept in the area of Multi-Agent Systems Learning.

1 Introduction

1.1 Framework

The objective of this work is to contribute to an answer to the question: "(How) can agents benefit from mutual interaction during the learning process, in order to achieve better individual and overall system performances?". This question has been deemed a "challenging issue" by several authors in recently published work (Sen, 1996; Weiß and Dillenbourgh, 1999; Kazakov and Kudenko, 2001; Matarić, 2001).

In the pursuit of an answer to this question, the objects of study are the interactions between the Learning Agents (hereafter referred as agents for the sake of simplicity) and the effects these interactions have on individual and global learning processes. Interactions that affect the learning process can take several forms in Multi-Agent Systems (MAS). These forms range from the indirect effects of other agents' actions (whether they are cooperative or competitive), to direct communication of complex knowledge structures, as well as cooperative negotiation of a search policy or solution.

The most promising way in which cooperative learning agents can benefit from interaction seems to be by exchanging (or sharing) information regarding the learning process itself. As observed by Tan (1993) agents can exchange information regarding several aspects of the learning process: a) the state of the environment, b) episodes (state, action, reward triplets), or c) internal parameters and policies.

Exchanging environment states can be seen as a form of shared exploration. Sharing this information may require a large amount of communication, although the use of a selective policy for the exchange of information may reduce this cost. This type of interaction may be seen as if each agent has extra sets of sensors spread out in the environment, being able to have a more complete view of its external state. This larger view of the state space may require either pre-acquired knowledge on how to interpret this information and integrate it with its own view of the environment's state, or simply be considered as extra input providing a wider range of information about the state. In the limit case, where all agents have access to information regarding the state sensed by all their peers, each agent could be seen as a classical Machine Learning (ML) system with distributed sensors if we consider other agents' actions as part of the environment. One interesting difference, though, is the fact that other agents sensors are not under the control of the learning agent and the perspective they provide on the world may be biased by the needs of the owner of the sensor.

Episode exchange requires that the agents are (or have been) facing similar problems, requiring similar solutions and may also lead to large amounts of communication if there is no criteria regulating the exchange of information. In the limit case, where all agents share all the episodes, this process can also be seen as a single learning system, and produce very little new knowledge. In fact, the exchange of too much data could lead all the agents to follow the same path through the search space, wasting valuable exploration resources.

Sharing internal parameters is another way in which agents can benefit from the knowledge obtained by their peers. Again, in the limit, this can be seen as the use of a single learning agent if communication is unrestricted. This type of information exchange requires that agents have similar internal structures, so that they can easily map their peers' internal parameters into their own, or that they share a complex domain ontology.

As can be seen in the above paragraphs the question is not only: "what type of information to exchange?", but also "when to exchange information?" and "how much information is it convenient to exchange?". When considering human cooperative learning in a team, a common method to improve one's skills is to ask for advice at critical times, or request a demonstration of a solution to a particular problem to someone who is reputed to have better skills in the subject. This is what we have attempted to translate into the realm of Multi-Agent Systems Learning (MASL). Another interesting outcome of our experiments concerns the degree of adequacy different algorithms, being used by the agents, exhibit for different situations considered in the scenario. However, this is not of great importance where MASL is concerned, except in which regards an agent's knowledge about whom to request an advice to in each particular situation. It is our hope that different agents specialise in different situations, becoming, up to a certain extent, complementary in the Multi-Agent System context.

1.2 Rationale and summarized description

This paper reports experiments in which agents selectively share episodes by requesting advice for given situations to other agents whose score is, currently, better than their own in solving a particular problem. Considering the discussion of the previous section, this option seemed the most promising for the following reasons:

a) Sharing of episodes does not put heavy restrictions on the heterogeneity of the underlying learning algorithms;
b) Having different algorithms solving similar problems may lead to different forms of exploration of the same search space, thus increasing the probability of finding a good solution;
c) It is more informative and less dependent on precoded knowledge than the exchange of environment's states.

Experiments were conducted with a group of Learning Agents embedded in a simplified simulation of a traffic control problem to test the advantages and problems of advice-exchange during learning. Each individual agent uses a standard version of a well know, sub-symbolic, learning algorithm (Random Walk, Evolutionary Algorithms, Simulated Annealing, and Q-Learning). Agents are heterogeneous (i.e., each applies a different learning mechanism, unknown to others). This fact makes communication of internal parameters or policies suffer from the above-mentioned disadvantages, thus it was not considered. The information exchanged amongst agents is: current state (as seen by the advisee agent); best response that can be provided to that state (by the advisor agent); present and best scores, broadcasted at the end of each training stage (epoch).

The problem chosen to test the use of advice-exchange has, as most problems studied in MASL, the following characteristics:

a) Analytical computation of the optimal actions is intractable;
b) The only information available to evaluate learning is a measure of the quality of the present state of the system;
c) The same action executed by a given agent may have different consequences at different times, even if the system is (as far as the agent is allowed to know) in the same state;
d) The agent has only a partial view of the problem's state.

The simplified traffic control problem chosen for these experiments requires that each agent learn to control the traffic-lights in one intersection under variable traffic conditions. Each intersection has four incoming, and four outgoing, lanes. One agent controls the four traffic lights necessary to discipline traffic in one intersection. In the experiments reported here, the crossings controlled by each of the agents are not connected.

The learning parameters of each agent are adapted using two different methods: a reinforcement-based algorithm, using a quality measure that is directly supplied by the environment, and supervised learning using the advice given by peers as the desired response. Notice that the term "reinforcement-based" is used to mean "based on a scalar quality/utility feedback", as opposed to supervised learning which requires a desired response as feedback. The common usage of the term "reinforcement learning", that refers to variations of temporal difference methods (Sutton and Barto, 1987), is a subclass of reinforcement-based algorithms, as are, for instance, most flavours of Evolutionary Algorithms.

2 Related Work

The advantages and drawbacks of sharing information and using external teachers in variants of Q-Learning (Watkins and Dayan, 1992) had some important contributions in the early 90's. Whitehead (1991) reports on the usage of two cooperative learning mechanisms: Learning with an External Critic (LEC) and Learning By Watching (LBW). The first, (LEC), is based on the use of an external automated critic, while the second (LBW), learns vicariously by watching other agent's behaviour (which is equivalent to sharing state, action, quality triplets). This work proves that the complexity of the search mechanisms of both LEC and LBW is inferior

to that of standard Q-Learning for an important class of state-spaces. Experiments reported in (Whitehead and Ballard, 1991) support these conclusions.

Lin (1992) uses a human teacher to improve the performance of two variants of Q-Learning. This work reports that the "advantages of teaching should become more relevant as the learning task gets more difficult". Results presented show that teaching does improve learning performance in the harder task tested (a variant of the maze problem), although it seems to have no effect on the performance on the easier task (an easier variant of the same maze problem).

The main reference on related work is (Tan, 1993). Tan addressed the problem of exchanging information during the learning process amongst Q-Learning agents. This work reports the results of sharing several types of information amongst several (Q-Learning) agents in the predator-prey problem. Experiments were conducted in which agents shared policies, episodes (state, action, quality triplets), and sensation (state). Although the experiments use solely Q-Learning in the predator-prey domain, the author believes that: "conclusions can be applied to cooperation among autonomous learning agents in general". Conclusions point out that "a) additional sensation from another agent is beneficial if it can be used efficiently, b) sharing learned policies or episodes among agents speeds up learning at the cost of communication, and c) for joint tasks, agents engaging in partnership can significantly outperform independent agents, although they may learn slowly in the beginning". Results presented in (Tan, 1993) also appear to point to the conclusion that sharing episodes with peers is beneficial and can lead to a performance similar to that obtained by sharing policies. Sharing episodes volunteered by an expert agent leads to the best scores in the presented tests, significantly outperforming all other agents in the experiments.

After these first, fundamental, works several variants of information sharing Q-Learners appeared reporting good results in the mixture of some form of teaching and reinforcement learning.

Baroglio (1995) uses an automatic teacher and a technique called "shaping" to teach a Reinforcement Learning algorithm the task of pole balancing. Shaping is defined as a relaxation of the evaluation of goal states in the beginning of training, and a tightening of those conditions in the end.

Clouse (1996) also uses an automatic expert trainer to give the agent actions to perform, thus reducing the exploration time.

Matarić (1996) reports on the use of localized communication to share sensory data and reward as a way to overcome hidden state and credit assignment problems in groups of agents. The experiments conducted in two robot problems, (block pushing and foraging) show improvements in performance on both cases. Later work by the same author, (Matarić, 2001) reports several good results using human teaching and learning by imitation in robot tasks. Experimental results can be found in (Jenkins et al. 2000; Nicolescu and Matarić, 2001; Matarić, 2001b).

Brafman and Tenemholtz (1996) use an expert agent to teach a student agent in a version of the "prisoner's dilemma". The agents implement variations of Q-Learning.

Maclin and Shavlik (1997) use human advice, encoded in rules, which are acquired in a programming language that was specially designed for this purpose. These rules are inserted in a Knowledge Based Neural Network (KBANN) used in Q-Learning to estimate the quality of a given action.

Berenji and Vengerov (2000) report analytical and experimental results concerning the cooperation of Q-Learning agents by sharing quality values amongst them. Experiments were conducted in two abstract problems. Results point out that limitations to cooperative learning described in (Whitehead, 1991) can be surpassed successfully under certain circumstances, leading to better results than the theoretical predictions foresaw.

Simultaneous uses of Evolutionary Algorithms (Holland, 1975; Koza, 1992) and Backpropagation (Rumelhart, Hinton and Williams 1986) are relatively common in Machine Learning (ML) literature, although in most cases Evolutionary Algorithms are used to select the topology or learning parameters, and not to update weights. Some examples can be found in (Salustowicz, 1995) and (Yao, 1999). There are also reports on the successful use of Evolutionary Algorithms and Backpropagation simultaneously for weight adaptation (Topchy, Lebedko and Miagkikh, 1996; Ku and Mak, 1997; Ehardh et al. 1998). Most of the problems in which a mixture of Evolutionary Algorithms and Backpropagation is used are supervised learning problems, i.e., problems for which the desired response of the system is known in advance (not the case of the problem studied in this paper). Castillo et al. (1998) obtained good results in several standard ML problems using Simulated Annealing and Backpropagation, in a similar way to that which is applied in this work. Again, this was used as an add-on to supervised learning to solve a problem for which there is a well known desired response.

The use of learning techniques for the control of traffic-lights can be found in (Goldman and Rosenschein, 1995; Thorpe, 1997; Brockfeld et al. 2001).

3 Experimental Setup

This section will describe the internal details of the traffic simulation, the learning mechanisms and the advice-exchange technique.

3.1 The Traffic Simulator

The traffic simulator environment is composed of lanes, lane-segments, traffic-lights (and the corresponding controlling agents), and cars. Cars are "well behaved", in the sense that they:

a) Can only move forward;
b) Do not cross yellow or red-lights;
c) Move at a constant speed;
d) Do not crash into other cars.

Cars are inserted at the beginning of each lane, whenever that space is empty, with a probability that varies in time according to a saw-tooth function, of the form:

$$pInsert(t) = (M - m)((t + T_0)\%T) / T + m \quad (1)$$

where T_0 is the initial delay, T is the period, m the minimum probability for car insertion and M the maximum. The parameters used in the experiments discussed here for the generation of new cars in the beginning of each lane were in the following ranges: $0<T_0\leq T$, $50\leq T\leq 5000$, $0.01\leq m\leq 0.1$, $0.01\leq M\leq 0.3$. In further experiments different generation functions were used, mostly based in superimposition of gaussian functions, but the results reported here were acquired using the saw-tooth generation-function.

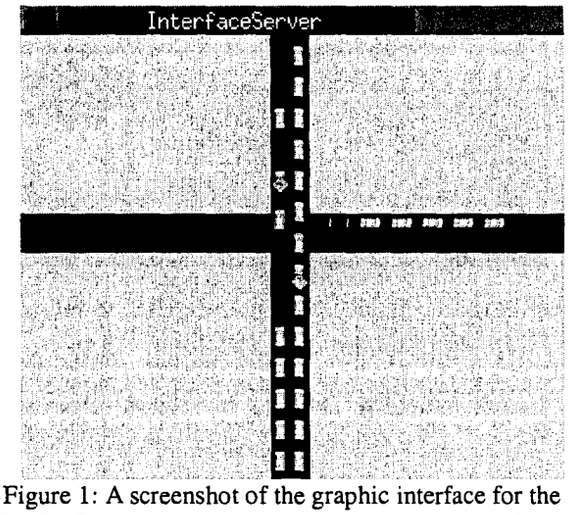


Figure 1: A screenshot of the graphic interface for the Traffic Simulator, showing a partial view of a local scenario.

The time-unit used throughout this description is one turn. One turn corresponds to a period where each object in the system is allowed to perform one action and all the necessary calculations for it. Lanes have three lane-segments: incoming (before the crossing, where cars are inserted), crossing and outgoing. Each local scenario (Figure 1) consists of four lanes, each with a different movement direction and one crossing (the lanes in a local scenario will be referred as North, South, East and West, for the remainder of this description). In the experiments reported here the local scenarios are not connected, i.e., each lane has only one crossing and one traffic light. Cars are inserted in its incoming lane-segment and removed when they reach the extremity of its outgoing lane-segment, after having passed the crossing. Each incoming lane-segment was designed to hold a maximum of 60 cars.

At the beginning of each green-yellow-red cycle, the agents observe the state of environment for their local scenario and decide on the percentage of green-time (g) to attribute to the North and South lanes (the percentage of time attributed to the East and West lanes is automatically set at $1 - g$. Yellow-time is fixed in each experiment and lies in the interval [10, 15] turns). Two types of description of the environment's state are used, the first is realistic in the sense that it is technically achievable to collect that type of data in a real situation and it is actually used by traffic controllers today. The second, although it may be unfeasible in today's traffic monitoring systems, was considered to have relevant information for the learning process.

In the first type of state representation, the state $s(t)$, at time t, is composed by four scalar values (s_N, s_S, s_E, s_W), where each component (s_i) represents the ratio of the number of incoming vehicles ($n_i(t)$) in lane i relative to the total number of incoming vehicles in all lanes. This state representation will be referred as *count state representation*.

$$s_i(t) = \frac{n_i(t)}{\sum_j n_j(t)} (i, j \in \{N, S, E, W\}) \quad (2)$$

The second type of environment state has the same information as the one described above plus four scalar values, each of which represents the lifetime (number of turns since creation) of the incoming vehicle that is closest to the traffic-light ($life\_first_i(t)$). To keep inputs within the interval [0,1], this value was cut-off at a maximum lifetime (*lifemax*), and divided by the same value. Thus, the four extra scalar values are:

$$s_i(t) = \frac{life\_first_i(t)}{life\ \max} (i, j \in \{N, S, E, W\}) \quad (3)$$

if $life\_first(t) < lifemax$ or 1 otherwise. The value of *lifemax* was chosen to be 3 to 10 times the number of turns a car takes to reach the crossing at average speed, depending on the difficulty of each particular scenario, which is mainly dependent on the parameters used for car generation. This state representation will be referred as *count-time state representation*. The state representations described above are similar to the ones that were reported to have produced some of the best results in the experiments conducted by Thorpe (1997) for the same type of problem (learning to control traffic-lights at an intersection). The normalization of the inputs to fit the [0,1] interval was necessary, even at the cost of loss of information, for two main reasons: a) it keeps the first layer of sigmoids from reaching saturation too early in the learning process; b) using percentages for the first four elements of the state space allows a substantial reduction of the number of possible states, as described below when the implementation of Q-Learning is discussed.

The quality of service of each traffic-light controller at time t, is given by $q(t)$, which was initially calculated according to

$$q(t) = 1 - \frac{\sum_i life_i(t) / n}{life\max}, \quad (4)$$

where $life_i(t)$ is the number of turns since creation of car i at time t and *lifemax* has the same meaning as above. The sum is made for all (n) cars in the incoming lane-segments of a crossing. This measure did not provide enough differentiation of "good" and "bad" environment states, thus a logistic function was introduced, using $q(t)$, in (4), as input, to emphasize the difference in quality between these two types of environment states. A comparative view of both functions can be seen in Figure 2, the former in continuous line the latter in dashed line style.

The car generation parameters in traffic simulator proved difficult to tune. Slight changes led to simulations that were either too difficult (no heuristic nor any learned strategy were able to prevent major traffic jams), or to problems in which both simple heuristics and learned strategies were able to keep a normal traffic flow with very few learning steps.

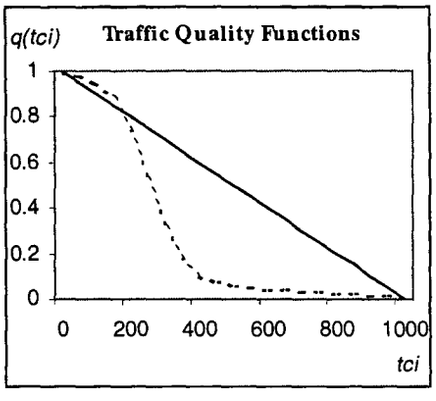


Figure 2: Two functions for the evaluation of traffic quality based on the average time of life of the incoming cars (*tci*).

The traffic simulator was coded in C++, with a Java graphical interface (Figure 1). Agents are not independent processes, at this stage they are merely C++ objects that are given a turn to execute their actions in round-robin. On the one hand, this choice eliminates the "noise" of asynchronous communication and synchronization of parallel threads, on the other hand, lighter agents that perform simple but coarse learning techniques (like Random Walk) are being slowed down by the more computationally intensive learning algorithms (like Q-Learning). This may prove an interesting ground to cover in future experiments. The real-time competition between fast and simple learning strategies against slower but more refined ones can have interesting consequences in the effect of advice-exchange.

Although this was not an issue, the simulation runs faster than real-time, even when all agents are performing learning steps. Simulations ran (usually) for 1600 epochs, where each epoch consists of 50 green-yellow-red cycles, each consisting of 100 turns in which, on average, approximately 150 cars were moved and checked for collisions. Each simulation, with five disconnected crossings (i.e., four parallel learning algorithms and one heuristic agent), took 4 to 5 hours to run in a Pentium IV at 1.5 GHz. To generate a set of comparable data, this scenario must be run twice: with and without advice-exchange.

3.2 Learning Agents

This section describes the learning algorithms used by each of the agents involved in the experiments, as well as the heuristic used for the fixed strategy agent.

3.2.1 Stand-alone agents

The stand-alone versions of the learning agents are used to provide results with which the performance of advice-exchanging agents could be compared. The stand-alone agents implement four classical learning algorithms: Random Walk (RL), which is a simple hill-climbing algorithm, Simulated Annealing (SA), Evolutionary Algorithms (EA) and Q-Learning (QL). A fifth agent was implemented (HEU) using a fixed heuristic policy. As the objective of these experiments was not to solve this problem in the most efficient way, but to evaluate advice-exchange for problems that have characteristics similar to this, the algorithms were not chosen or fine-tuned to produce the best possible results for traffic control. The choice of algorithms and their parameters was guided by the goal of comparing the performance of a heterogeneous group of learning agents, using classical learning strategies, in a non-deterministic, non-supervised, partially-observable problem, with and without advice-exchange.

All agents, except QL and HEU, adapt the weights of a small, one hidden layer, neural network. Experiments were conducted with several topologies, but the results discussed below refer to fully connected networks of 4x4x1, when using *count state representation*, and 8x4x1, when using *count-time state representation*. The weights of these networks were initialised randomly with values in the range [-0.5, 0.5]. The hidden layer is composed of sigmoids whose output varies in [-1, 1], while the outer layer sigmoids' output is in the range [0, 1]. This neural network will produce an output that will be the percentage of green-time (*gt*) for the next green-yellow-red cycle.

The Random Walk (RW) algorithm simply disturbs the current values of the weights of the neural network by adding a random value in the range [$-d$, d], where d is the maximum disturbance, which will be updated after a given number of epochs, (*ne*), according to $d=\gamma d$, with $0<\gamma<1$, until it reaches a minimum value, (*min_d*). An epoch consists of n green-red-yellow cycles. At the end of an epoch, the new set of parameters is kept if the av-

erage quality of service in the controlled crossing during that epoch is better than the best average quality achieved so far. The values used for the parameters of this algorithm in the experiments discussed here were in the following intervals: $d \in [0.5, 0.7]$, min_d=0.01, γ=0.99, n=50, $n \in [3, 7]$. These values apply also for the decay of disturbance limits in the following descriptions of SA and EA. When referring to the intervals in which values were chosen, it is meant that in different experiments several combinations of parameter values were tested but the initial value for these parameters was always in the mentioned range.

Simulated Annealing (SA), (Kirkpatrick, Gelatt and Vecchi, 1983), works in a similar way to Random Walk, but it may accept the new parameters even if the quality has diminished. New parameters are accepted if a uniformly generated random number $p \in [0,1[$, is smaller than

$$pa(t) = e^{-\Delta q / T}, \quad (5)$$

where T is a temperature parameter that is decreased during training in the same way as d in RW and Δq is the difference between the best average quality achieved so far and the average quality of the last epoch.

Evolutionary Algorithms (EA), (Holland, 1975;Koza, 92), were implemented in a similar way to the one described in (Glickman and Sycara, 1999), which is reported to have been successful in learning to navigate in a difficult variation of the maze problem by updating the weights of a small Recurrent Artificial Neural Network. This implementation relies almost totally in the mutation of the weights, in a way similar to the one used for the disturbance of weights described for RW and SA. Each set of parameters (specimen), which comprises all the weights of a neural network of the appropriate size for the state representation being used, is evaluated during one epoch. After the whole population is evaluated, the best n specimens are chosen for mutation and recombination. An elitist strategy is used by keeping the best b specimens untouched for the next generation. The remainder of the population is built as follows: the first m are mutated, the remaining specimens (r) are created from pairs of the selected specimens, by choosing randomly from each of them entire layers of neural network weights. The values used for the parameters of this algorithm in the experiments discussed here were in the following intervals: $n \in [7, 10]$, $b \in [3, 7]$, $m \in [15, 25]$, $r \in [2, 5]$. The size of the population was [20, 30].

Q-Learning (QL), (Watkins and Dayan 1992), uses a lookup table with an entry for each state-action pair in which the expected utility $Q(s,a)$ is saved. $Q(s,a)$ represents the expected utility of doing action a when the environment is in state s. Utility is updated in the usual way, i.e.,

$$Q(s,a) = Q(s,a) + \alpha(r + \beta Q \max(s') - Q(s,a)), \quad (6)$$

where s' is the state after performing action a, α is the learning rate, β the discount factor and $Q_{max}(s)$ is given by

$$Q_{\max}(s) = \max_a (Q(s,a)), \quad (7)$$

for all possible actions a when the system is in state s.

The values of α (learning rate), in the different experiments, were in the interval [0.5,0.7]. The learning rate is updated after a given number of epochs, (ne), according to $\alpha = \gamma \alpha$, with $0<\gamma<1$, until it reaches a minimum value (which in this case was 0.012). In the experiments discussed here ne=5. Parameter β (discount) was fixed in each experiment. Different values for β were tested in several experiments within the interval [0.6,0.8]. The choice of action a, given that the system is in state s, was done with probability $p(a|s)$ that is given by a Boltzman distribution

$$p(a \mid s) = \frac{e^{Q(s,a)/T}}{\sum_i e^{Q(s,a_i)/T}}, \quad (8)$$

where T is a temperature parameter whose initial value was in the interval [0.3,0.7] and was decayed in a similar way to the one described for α.

Since the state of the environment is a real-valued vector, a partition of the space in a square lattice is required to map environment states (continuous) to internal (discrete) states. The decision of which is the state of the environment at a given time is made by calculating the Euclidean distance between the continuous valued world state and each of the discrete state representations and selecting the state with minimum distance. For the *count state representation* this partition consists in states composed of quadruples of the form: (x_1, x_2, x_3, x_4), for which $x_1 + x_2 + x_3 + x_4 = 1.0$, and $x_i \in \{0, 0.1, 0.2, \ldots, 0.9, 1.0\}$. This reduction of the state space, compared to the use of all possible quadruples with elements in {0.0, 0.1, 0.2,...}, is possible given that the representation of the environment is composed of the percentages of vehicles in each lane relative to the number of vehicles in all lanes, thus being restricted to quadruples for which the sum of all elements is 1.0. For the *count-time state representation* the internal state is of the form: (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8), where the first four parameters are generated in the same fashion as in the previous case but with a coarser granularity and, the last four elements, are selected combinations of values in {0.0, 0.25, 0.5, 0.75, 1.0}. The number of states for the first and second case is, respectively, 286 and 1225. In future experiments, with more informative state-space representations, it may become necessary to use a neural network to map states to their correspondent utility as described in (Barto, Sutton and Watkins, 1990; Lin, 1992). Actions, i.e., green-time for the North and South lanes, are also considered as discrete values starting from zero, up to the maximum green time allowed, and differing by 0.05 steps.

The heuristic agent (HEU) gives a response that is calculated in different ways, depending on the state representation. The percentage of green-time (g) for the North and South lanes is calculated by

$$g = \frac{\max(n_N, n_S)}{\max(n_N, n_S) + \max(n_E, n_W)} \quad (9)$$

for the *count state representation*, and in a similar way accounting for the *lifetime* values for the first car in each track for the *count-time state representation*. The idea is that it seems reasonable to attribute a green time to the North and South lanes proportionally to the magnitude of the maximum number of cars (and the waiting times) relative to that of the sum of these values for both pairs of lanes.

3.2.2 Advice-exchange mechanism

The main expectation, when advice-exchange was chosen, was that using advice from the more knowledgeable agents in the system would improve the learning performances of all agents. Since supervision is a more efficient training method than reinforcement, (at the expense of needing more information) then, when no supervision information is available why not use advice as supervision? Better yet, if agents have different learning skills, which produce different types of progress through the search-space, they may be able to avoid that others get stuck in local minima by exchanging advice. It is unlikely that all agents are stuck in the same local minima and the exchange of information regarding the appropriate answers to some environment states could force others to seek better solutions.

The process of advice-exchange is conducted in a different way in the agents that use a neural network as activation function and in the Q-Learning agent. The heuristic agent does not participate in the experiments concerning advice-exchange. Advice-exchange is prohibited in the first 2 to 10 epochs of training, depending on the experiments, to avoid random advice being exchanged and to allow some time for the agents to announce a credible best average quality value. All agents broadcast their best result (i.e., best average quality measured during one epoch) at the beginning of each epoch.

At the beginning of each green-yellow-red cycle, agent i (the advisee) evaluates its current average quality (cq_i) since the beginning of the present epoch. This quality is compared with the best average quality (bq_j), for all agents j, broadcasted by other agents at the end of last epoch. Let $mbq_k = max(bq_j)$, for all agents $j \neq i$. If $cq_i < d\ mbq_k$ where d is a discount factor (usually 0.8), then agent i will request advice from agent k (the advisor) who as advertised the best average quality. The request for advice is sent having as parameter the current state of the environment as seen by agent i. The advisor switches his working parameters (neural network weights in most cases) to the set of parameters that was used in the epoch where the best average quality was achieved and runs the state communicated by the advisee producing its best guess at what would be the appropriate response to this state. This response (the advised percentage of green time for the north and south lanes) is communicated back to the advisee. In the case where advisees are RW, SA and EA agents, the communicated result is backpropagated as desired response, using the standard backpropagation rule (Rumelhart, Hinton and Wlliams, 1986) to update the weights of the network immediately after this pass. In some experiments an adaptive learning rate backpropagation (Silva and Almeida, 1990) was used but results were not significantly different. The values for the main backpropagation parameters used in the experiments discussed here were in the following intervals: learning rate: [0.001, 0.05], momentum: [0.3, 0.7].

Table 1: Steps of the advice-exchange sequence for an advisee agent (i) and an advisor agent (k).

1. Agent i: receive the best average quality (bq_j) from all other agents ($j \neq i$). Quality for Agent i is cq_i.
2. Agent i: get state s for evaluation.
3. Agent i: calculate $k = \arg \max_j(bq_j)$, for all agents ($j \neq i$).
4. Agent i: if $cq_i < d\ max(bq_j)$:
   a. Agent i: send agent k the current state s and request advice.
   b. Agent k: switch to best parameters and run state s to produce its best guess at the adequate response (g).
   c. Agent k: return g to Agent i.
   d. Agent i: process advice (g).
5. Agent i: run state s and produce response g'.

When the Q-Learning agent is the advisor, switching to best parameters corresponds simply in selecting the action with best quality. In the case where the Q-Learning agent is the advisee, the action that is closest to the given advice (recall that actions are discrete values in this case) is rewarded in a similar way to that described in (6). Since in this case the state of the system after action a is unknown, the value of $Q_{max}(s')$ is replaced by a weighted average of the utilities of all the possible following states when executing action a at state s:

$$Q_{\max}(a) = p(s' \mid a, s) Q_{\max}(s') \quad (10)$$

where $p(s'|a,s)$ is the probability of a transition to state s' given that action a is executed at state s and it is calculated based on previous experience, as the number of transitions ($nt_{s'as}$) to state s' when performing action a at the current state, s, relative to the total number of transitions from current state by action a, i.e.,

$$p(s' \mid a, s) = \frac{nt_{s'as}}{\sum_i nt_{ias}}, i \in S_{sa} \quad (11)$$

where S_{sa} is the set of states reachable from state s by action a. This type of adaptation of the state utility was proposed in Sutton's (1992) Dyna.

After updating the internal parameters with the advised information, the advisee agent gives the appropriate response to the system following the normal procedure for each particular algorithm.

4 Experimental Results

Before the discussion of the experimental results, let us put forward a few brief remarks concerning the simulation and experiments.

The type of problem we are dealing with is a difficult topic for simulation. Several works have been done in this area, and the simplifications made in this scenario were, in great measure, inspired by previous works mentioned in section 2. Nevertheless, the tuning of the simulator for the problem at hand was not a trivial matter. The problems tended to be either too easy or too hard, and, in the first experiments, only marginal differences could be observed in the quality measure during training. The most interesting experiments conducted were the cases where lanes had quite different behaviours from one another, ranging from a medium steady flow, to high peaks of traffic intermediated with periods with nearly no traffic at all. As observed in (Lin 1992), when doing similar experiments with variants of Q-Leaning, harder tests provided the best results for advice-exchange. However, there seems to be a fine line between hard solvable problems and, apparently, insoluble tasks in which no learning strategy, nor heuristic, could reach reasonable values of quality.

The interpretation of results is also not an easy task. The fact that agents are running online, and most of them are based on random disturbance, added to the stochastic nature of the environment, produces very "noisy" quality evaluations. The results presented here focus mainly on the analysis of the evolution of the best quality achieved up the present moment of training. Other measures also give us an insight on the process, but, at the present moment this seemed to be the one that could better illustrate the main observations made during experiments.

The above-mentioned stochastic nature of the problem, and the large simulation times, also forced a compromise in the choice of parameters for car generation. Although an even greater variety of behaviours could be achieved with other type of functions, whose periods span over a larger time-frame, this would require that each training epoch would be much longer, so that a comparison between values of different epochs would be fair. A lot of care was put into making epochs equally hard, in terms of frequency of cars generated.

One last remark concerning the discussion of results that will follow. The amount data necessary for a sound statistical comparison and evaluation of this technique is still being gathered. The preliminary results discussed here, produced in a series of 30 full trials, give us an insight on the problems and possible advantages of advice-exchange during learning, but data is still not sufficient for a detailed evaluation of the advantages and drawbacks of this technique. The above mentioned trials were ran under different conditions, either in the parameters of car-generation, lane-size and car speeds, or in the decay rates and other parameters of the algorithms themselves.

Before starting experiments, some results were expected, namely:

a) Initial disturbance of the learning process due to advice by non-expert peers, as reported by Tan (1993) for cooperation amongst Q-Learning agents.
b) After a few epochs, fast, step-like, increases in quality of response, as soon as one of the agents, found a better area of the state space and drove other agents that had poorer performances to that area.
c) Final convergence on better quality values than in tests where no advice is exchanged.
d) Problems of convergence when using excess of advice, or high learning rates when processing advice.
e) Improved resistance to bad parameterisation (special in algorithms like Simulated Annealing, which have parameters, like temperature, that are difficult to tune).

The actual observed results differed in some respects from expectations. The initial disturbance, or slower convergence, reported by Tan (1993) for Q-Learning agents, was not observed as a rule, although it occasionally happened. The exact opposite was also observed. In some experiments we can find agents that use advice climbing much faster to a reasonable quality plateau. Occasionally learning was much slower afterwards (probably a local maximum was reached) and this high initial quality value was gradually surpassed by the stand-alone algorithms during the rest of the training.

The second expectation, the appearance of high steps in the quality measure, due to advice from an agent that discovered a much better area of the search-space, was observed, but seems to be far less common than expected. Figure 3 shows a detail of the initial phase of a trial where we can see a typical situation of the described behaviour. The Simulated Annealing agent jumps to a high quality area, and "pulls" Random Walk and Q-Learning into that area in a few epochs. In this experiment the advice-exchanging algorithms did not stop at this quality plateau, being able to obtain better scores than their counterparts.

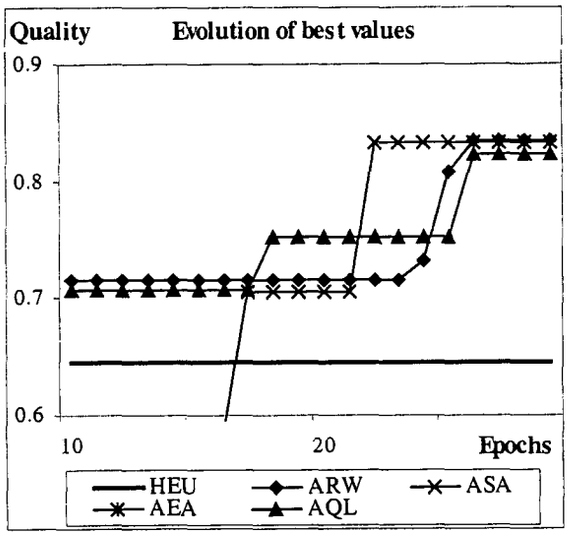


Figure 3: Detail of the initial phase of a trial where advice given by Simulated Annealing (ASA) led Random Walk (ARW) and Q-Learning (AQL) agents on a sudden climb of more than 10%. Evolutionary Algorithms also benefited from this jump, but the climb was less steep and from a lower point.

Results where the final quality values for the best agent, on trials with advice-exchange, is significantly better than in the normal case were observed, but do not seem to be as common as expected. Figures 4 to 7 show comparisons of the methods with and without advice-exchange for one of the trials where advice-exchange proved advantageous. Notice that all results are better than the one obtained by the heuristic agent (HEU), which was not frequent and denotes a particularly hard problem. The most usual result is that agents climb to the vicinity of the best agent's quality in few epochs, and make only minor improvements for rest of the trial.

The expectations referred in d) and e) were observed, as was foreseen. In fact, several cases were observed in trials without advice-exchange, where early freezing of the temperature parameter or the decay of the exploration rate, led to a sudden drop to a low-quality valley, from which the algorithm did not escape for the rest of trial. These events are rare in trials using advice-exchange.

One of the most interesting problems observed was that of ill advice. It was observed that some agents, due to a "lucky" initialisation and exploration sequence, never experience very heavy traffic conditions, thus, their best parameters are not suited to deal with this problem. When asked for advice regarding a heavy traffic situation, their advice is not only useless, but harmful, because it is stamped with the "quality" of an expert. In Q-Learning this was easy to observe because there were situations, far into the trials, for which advice was being given concerning states that had never been visited before. In the next section some measures to prevent this problem will be discussed.

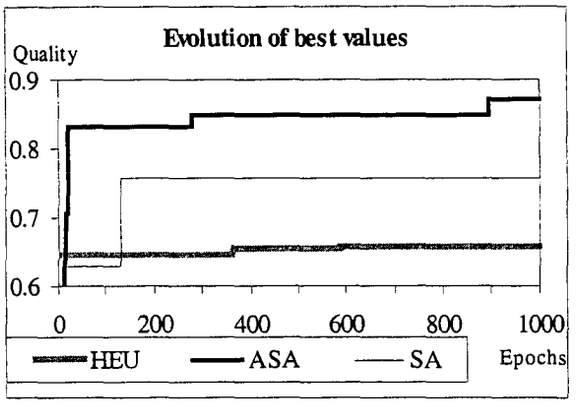


Figure 4: Comparison of Simulated Annealing performance, with (ASA) and without (SA) advice-exchange, and the corresponding heuristic (HEU) quality for the same trial.

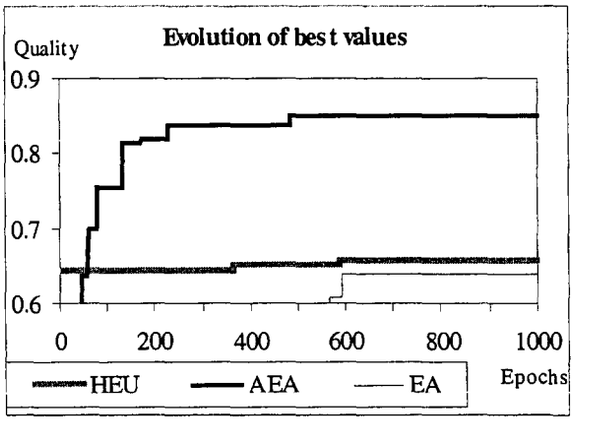


Figure 5: Comparison of Evolutionary Algorithms performance, with (AEA) and without (EA) advice-exchange, and the corresponding heuristic (HEU) quality for the same trial.

5 Conclusions and Future Work

As mentioned in the previous section, advice-exchange seems to be a promising way in which agents can profit from mutual interaction during the learning process. However, this is just the beginning of a search, where a few questions were answered and many were raised. A thorough analysis of the conditions in which this technique is advantageous is still necessary. It is important to discover how this technique performs when agents are not just communicating information about similar learning problems, but attempting to solve the same problem in a common environment. The application of similar methods to other type of learning agents, as well as other problems, is also an important step in the validation of this approach.

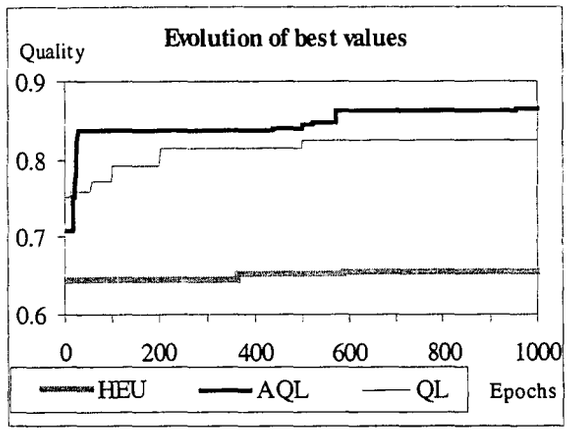


Figure 6: Comparison of Q-Learning performance, with (AQL) and without (QL) advice-exchange, and the corresponding heuristic (HEU) quality for the same trial

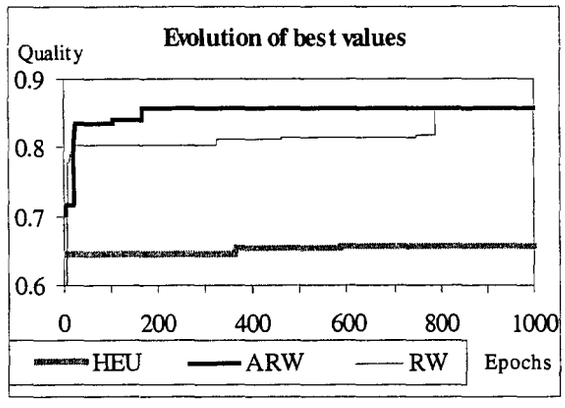


Figure 7: Comparison of Random Walk performance, with (ARW) and without (RW) advice-exchange, and the corresponding heuristic (HEU) quality for the same trial

For the time being, a more realistic traffic environment is under development based on the Nagel-Schreckenberg model for traffic simulation (Nagel and Shreckenberg, 1992). We hope that this new formulation provides a richer environment in which advice-exchange can be more thoroughly tested.

One of the main problems observed with advice-exchange is that bad advice, or blind reliance, can hinder the learning process, sometimes beyond recovery. One of the major hopes to deal with this problem is to develop a technique in which advisors can measure the quality of their own advice, and advisees can develop trust relationships, which would provide a way to filter bad advice. This may be especially interesting if trust can be associated with agent-situation pairs. This will allow the advisee to differentiate who is the expert on the particular situation it is facing. Work on "trust" has been reported recently in several publications, one of the most interesting for the related subject being (Sen, Biswas and Debnath, 2000).

Another interesting issue rises from the fact that humans usually offer unrequested advice for limit situations. Either great new discoveries or actions that may be harmful for the advisee seem to be of paramount importance in the use of advice. Rendering unrequested advice at critical points, by showing episodes of limit situations, also seems like a promising approach to improve the skills of a group of learning agents. The same applies to the combination of advice from several sources. These techniques may require an extra level of skills: more elaborate communication and planning capabilities, long-term memory, etc. These capabilities fall more into the realm of symbolic systems. The connection between symbolic and sub-symbolic layers, which has been also an interesting and rich topic of research in recent years, may play an important role in taking full advantage of some of the concepts outlined in this work. Our major aim is to, through a set of experiments, derive some principles and laws under which learning in the multi-agent system framework proves to be more effective, and inherently different from just having agents learning as individuals (even if they are together in the same environment).

Acknowledgements

The authors would like to thank, Manuel Sequeira, Thibault Langlois, Jorge Louçã , Pedro Figueiredo, Ana Violante, Rui Lopes, Ricardo Ribeiro, Francisco Pires, Isabel Machado, Sofia Regojo and two anonymous reviewers. Also, our thanks to the ResearchIndex crew.

References

C. Baroglio. Teaching by shaping. Proceedings ICML-95, Workshop on Learning by Induction vs. Learning by Demonstration, Tahoe City, CA, USA, 1995

A. G. Barto, R. S. Sutton and P. S. Brouwer. Associative search network: a reinforcement learning associative memory. Biological Cybernetics, 40(3):201-211, 1981

A. G. Barto, R. S. Sutton, C. J. C. H. Watkins. Learning and sequencial decision making. Garb & J. W. Moore (Eds.), Learning and computational neuroscience. MIT Press, 1990

H. R. Berenji and D. Vengerov. Advantages of Cooperation Between Reinforcement Learning Agents in Difficult Stochastic Problems. 9th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE '00), 2000

R. I. Brafman and M. Tennenholtz. On partially controlled multi-agent systems. Journal of Artificial Intelligence Research, 4:477-507, 1996

E. Brockfeld et al. Optimizing Traffic Lights in a Cellular Automaton Model for City Traffic. Physical Review E 64 , 2001

P. A. Castillo et al. SA-Prop: Optimization of Multilayer Perceptron Parameters using Simulated Annealing. IWANN99, 1998

J. A. Clouse. Learning from an automated training agent. Gerhard Weiß and Sandip Sen, editors, Adaptation and Learning in Multiagent Systems, Springer Verlag, Berlin, 1996

W. Ehardh et al. The Improvement and Comparison of different Algorithms for Optimizing Neural Networks on the MasPar {MP}-2. Neural Computation {NC}'98, ICSC Academic Press, Ed.M. Heiss, 617-623, 1998

M. Glickman and K. Sycara. Evolution of Goal-Directed Behavior Using Limited Information in a Complex Environment GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, July 1999

C. Goldman and J. Rosenschein. Mutually supervised learning in multi-agent systems. Proceedings of the IJCAI-95 Workshop on Adaptation and Learning in Multi-Agent Systems, Montreal, CA., August 1995

J. H. Holland. Adaptation in Natural and Artificial Systems. University of Michigan Press, 1975

O. C. Jenkins, M. J. Matarić and S. Weber. Primitive-based movement classification for humanoid imitation. Proceedings, first IEEE-RAS international conference on humanoid robotics, Cambridge, MA, MIT, 2000

D. Kazakov and D. Kudenko. Machine Learning and Inductive Logic Programmimng for Multi-Agent Systems. Multi Agents Systems and Applications: 9th EACCAI advanced course, Selected Tutorial Papers, 246-271, Prague, Czech Republic, July 2001

S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi. Optimization by simulated Annealing. Science, Vol. 220:671-680, May 1983

J. R. Koza. Genetic programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge MA, 1992

K. W. C. Ku and M. W. Mak. Exploring the effects of Lamarckian and Baldwinian learning in evolving recurrent neural networks. Proceedings of the IEEE International Conference on Evolutionary Computation, 617-621, 1997.

L.-J. Lin. "Programming Robots Using Reinforcement Learning and Teaching. Proceedings of the American Association for Artificial Intelligence (AAAI-91), 781-786, 1991

L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine Learning 8:293-321, 1992

R. Maclin and J. Shavlik. Creating advicetaking reinforcement learners. Machine Learning 22:251-281, 1997

M. J. Matarić. Using Communication to Reduce Locality in Distributed Multi-agent learning. Brandeis University Computer Science Technical Report CS-96-190, 1996

M. J. Matarić. Learning in behaviour-based multi-robot sysytems: policies, models and other agents. Journal of Cognitive Systems Research 2:81-93, Elsvier, 2001

M. J. Matarić. Sensory-motor primitives as a basis for imitation: linking perception to action and biology to robotics. C. Nehaniv & K. Dautenhahn (Eds.), Imitation in animals and artifacts, MIT Press, 2001b

K. Nagel, M Shreckenberg. A Cellular Automaton Model for Freeway Traffic. J. Phisique I, 2(12):2221-2229, 1992

M. Nicoluescu and M. J. Matarić. Learning and interacting in human-robot domains. K. Dautenhahn (Ed.), IEEE Transactions on systems, Man Cybernetics, special issue on Socially Intelligent Agents – The Human In The Loop, 2001

E. Oliveira, J.M.Fonseca, N. Jennings. Learning to be competitive in the Market. AAAI'99 - American Association of Artificial Intelligence Workshop on Negotiation, Orlando, USA, 1999

D. E. Rumelhart, G. E. Hinton and R. J. Wlliams. Learning internal representations by error propagation. Parallel Distributed Processing: Exploration in the Microstructure of Cognition, vol. 1: Foundations, 318-362, Cambridge MA: MIT Press, 1986

R. Salustowicz. A Genetic Algorithm for the Topological Optimization of Neural Networks. PhD Thesis, Tech. Univ. Berlin, 1995

S. Sen. Reciprocity: a foundational principle for promoting cooperative behavior among self-interested agents. Proc. of the Second International Conference on Multiagent Systems, 322-329, AAAI Press, Menlo Park, CA, 1996

S. Sen, A. Biswas, S. Debnath. Believing others: Pros and Cons. Proceedings of the Fourth International Conference on Multiagent Systems, 279-286, 2000

F. M. Silva and L. B. Almeida. Speeding up backpropagation. Advanced Neural Computers, 151-158, A'dam North-Holland, 1990

R. S. Sutton and A. G. Barto A Temporal-Difference Model of Classical Conditioning. Tech Report GTE Labs. TR87-509.2, 1987

R. S. Sutton. Reinforcement learning architectures. Proceedings ISKIT'92 International Symposium on Neural Information Processing, Fukuoka, Japan, 1992.

M. Tan. Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. Proceedings of the Tenth International Conference on Machine Learning, Amherst, MA, 330-337, 1993

T. Thorpe. Vehicle Traffic Light Control Using SARSA, Masters Thesis, Department of Computer Science, Colorado State University, 1997

A.P. Topchy, O.A. Lebedko and V.V. Miagkikh. Fast learning in multilayered neural networks by means of hybrid evolutionary and gradient algorithms Proc. of IC on Evolutionary Computation and Its Applications, Moscow, 1996

C. J. C H. Watkins and P. D. Dayan. Technical note: Q-learning. Machine Learning 8, 3:279-292, Kluwer Academic publishers, 1992

G. Weiß, P. Dillenbourg. What is 'multi' in multiagent learning. P. Dillenbourg (Ed.), Collaborative learning. Cognitive and computational approaches (Chapter 4, 64-80, Pergamon Press, 1999

S. D. Whitehead. A complexity Analisys of Cooperative Mechanisms in Reinforcement Learning. Proc. of the 9th National Conference on Artificial Inteligence (AAAI-91), 607-613, 1991

S. D. Whitehead and D. H. Ballard. A study of cooperative mechanisms for faster reinforcement learning. TR 365, Computer Science Department, University of Rochester, 1991

X. Yao. Evolving artificial neural networks. Proceedings of the IEEE, 87(9),1423-1447, 1999

AN AGENT ARCHITECTURE TO DESIGN SELF-ORGANIZING COLLECTIVES: PRINCIPLES AND APPLICATION

Gauthier Picard ; Marie-Pierre Gleizes
IRIT, Université Paul Sabatier
118, Route de Narbonne, 31062 TOULOUSE Cedex, France
0033 – 5.61.55.82.95
picard@irit.fr ; gleizes@irit.fr

Abstract

Designing collectives, which have a task to execute in a very dynamic environment, is a complex problem. Determining the right organization of these collectives in using group or role notions might be very difficult and even impossible for human analysts. Even if the organization can be found, it becomes not very easy to design entities, or agents in our case, which are able to take into account at the conception and design phases all possible situations an agent could face up to. Emergent and self-organizing approaches to model adaptive multi-agent systems avoid these difficulties. In this paper, we propose a new approach to design Adaptive Multi-Agent Systems with emergent functionality. This approach enables us to focus only on the design of the agents that compose the system. In fact, the self-organization of the system is led by the environmental feedback that each agent perceives. The interactions and the organization evolve, providing an adequate function to the system, which becomes adapted to its environment as well. Such functions have enough properties to be considered as emergent phenomena. First, we briefly present the Adaptive Multi-Agent Systems theory (AMAS) and its link with self-organization. In a second part, a multi-level architecture is proposed to model agents and to consider groups of agents as self-organizing collectives. In a third part, we describe a sample robot group behavior, the setting up of traffic in a constrained environment. Our architecture allows the emergence of a coherent collective behaviour: the dedication of corridors to specific directions. Finally, we show what is emergent by the analysis of results arising from measurements on collective phenomena.

1 Introduction

Nowadays Multi-Agent Systems (MAS) tackle complex and non-linear problem solving that classic systems are not able to resolve efficiently. These problems, such as flood prediction (Sontheimer et al., 2001), on-line brokerage (Athanassiou et al., 1999) or equation system solving, have been the SMAC (Cooperative Multi-Agent Systems) team's leitmotiv to elaborate new models based on self-organization. Robotics meets similar problems. The difficulty of understanding or of designing spatial settling, dangerous area minesweeping or resource transportation, becomes too high to elaborate single robots, which are able to resolve such tasks. That is how Collective Robotics is born, inspired by social insects communities like bees or ants like Kube and Zhang (1992) have highlighted.

High-level problem resolution by low-level entities interactions and the appearance of new functionality within groups are also two of the motivations of the Adaptive Multi-Agent Systems theory (AMAS). It takes part in a movement, named Emergence, which motivated many researchers around the world. Many groups have been formed to study this ill-known concept, which appears in the early Antiquity as Ali and Zimmer (1998) emphasize. Emergence was early quoted in computer science, notably by Holland (1998), and is linked to the Complexity Theory that tends to demystify this concept. Goldstein (1999) proposes a set of properties for the emergent systems such as radical novelty, coherence, macro and micro-level, dynamical and ostensive phenomena. These characterizations lead to new methodologies to design systems where the macro-level algorithm (the task of the system or the global function) does not appear at the micro-level (the agents' level). In Collective Robotics domain, Labbani-Igbida et al. (1998) propose such a methodology, named CIRTA. Our work takes place at this crossroads between MAS, Robotics and Emergence.

The MAS we are working on are self-organizing adaptive systems consisting in several autonomous agents situated in a common environment. Moreover, these agents participate to a coherent collective activity, to a common task. Because agent self-organization ability, they realize the global function of the system. The particularity is in the fact that we do not code within an agent the global function of the system. Because of the agents' self-organization ability, the system can adapt itself and the global function can emerge. The function realized by the system evolves when the organization of the agents in the system changes. We would like to show the pertinence of the Adaptive Multi-Agent Systems applied to Collective Robotics. Firstly, we present the AMAS theory on

which all developments are based. Then we describe our architecture. It is a generic model for our agents guided by the wish to apply cooperative interactions and self-organization. But a theory is nothing without its own application. So, secondly, a sample application is presented. It is about the study of the traffic of numerous robots in a constrained environment, composed of halls and narrow corridors like Vaughan et al. (2000a, b). The task is resource transportation between two halls. The question of learning and memorization will be raised. We distinguish several modules for different means to learn.

Our system has been simulated and results have been obtained and analysed. Finally, we conclude on the scope of our study and the perspectives of our work.

2 A Brief Overview of the AMAS Theory

In the AMAS theory, we propose to consider the equivalence between the organization of the collective and the global function obtained by the set of interactions between the low-level entities. To guaranty the emergent property of the global functionality, the collective has to be self-organizing.

2.1 Motivations

Several applications require the development of software that is characterized by an incomplete specification phase, because of the following reasons:

- the system has to evolve in a dynamical environment and it is impossible to totally specify all the situations the system may encounter;
- the system is open;
- the system is complex;
- there is no known algorithmic solution to resolve the problem;
- the internal organization of the system is a priori unknown;

The unexpected is inherent to this systems' life. Self-organization, which corresponds to an autonomously decided change, becomes a mean to overcome possible perturbations of the environment (MARCIA Group, 1996). This is a mean to realize adaptive systems too. In our systems the organization is treated as a result and not as a characteristic of the system to specify.

2.2 Definition and Characteristics

The AMAS theory is based on the self-organization by cooperation. In this theory:

Definition 1. *We call **adaptive multi-agent system** a multi-agent system which is able to change its behavior to adjust itself in its dynamical environment, either to realize the task it is intended to complete or to improve its functionality or performance. An adaptive multi-agent system is characterized by the following points:*

- *the system is plunged in a dynamical environment;*
- *the system realizes a function;*
- *the system is composed of interacting autonomous agents;*
- *each agent of the system realize a partial function;*
- *the organization of the system determines the result of the system;*

Learning, in the system point of view, consists on transforming its current function to adapt itself to its environment, i.e. changing its internal organization. So, this learning enables the system to have a right activity[1] in the environment where it is plunged into: this is the definition of the *functional adequacy*. So, the question is "when and how the system can transform itself to tend to the functional adequacy". The AMAS theory (Gleizes et al., 1999) says:

Theorem 1. *For all functionally adequate system, there is at least one cooperative system which realizes an equivalent function in the same environment.*

This result is important because it enables the guidance of the adaptive multi-agent system design. A first step lies in the identification of the agents and then to guarantee that each agent is or tends to be in cooperative interaction with the other agents. This method ensures the functional adequacy according to the theory.

In the AMAS theory, an agent is generally supplied with skills, communication and interaction (with other agents and/or the environment) capacities, beliefs and knowledge on other agents of the environment, aptitudes which enable the agent to reason, and a cooperation-based social attitude. The behaviour of each agent is specified in order to try to reach its objective(s) and to keep cooperative interactions with the other agents. Before any action, an agent locally examines if it is in cooperative interaction or not. In fact, it detects if it is in non cooperative situation. If the agent is in such a non cooperative situation, it tries to escape from this situation to return to a cooperative one. The cooperation is the social attitude that leads the behaviour of each agent by taking into account local criteria. This is what we call cooperation-based social attitude.

Therefore, an agent has two essential roles: the first is to realize its partial function. The second one is to act on the internal organization of the system. If an agent detects a non cooperative situation to return to a cooperative situation so as the system returns to a functionally adequate organization.

[1] The right activity is decidable only by an external observer who appreciates the interactions and who knows the function the system has to realize in its environment.

2.3 Non Cooperative Situations (NCS)

Agents, who are designed using the AMAS theory (Gleizes et al., 1999) and the associated methodology ADELFE (Bernon et al., 2001), have to respond to unexpected events. After identifying the agents, according to the AMAS theory, designers have to give to an autonomous agent the means to decide to change its interactions with the other agents. As we previously say, the change of the organization between the agents changes the function realized by the whole system. The means to self-organize is local to the agent. It consists in the ability to detect and to remove (if the agent can) all non cooperative interactions and to perform cooperative action when it is possible. There are three categories of non cooperative interactions:

- *Misunderstanding*: when a signal that is received from its environment can not be understood without ambiguity;
- *Incompetence*: when an information (an interpreted signal) cannot lead to new logical consequence. In other words, an information must bring novelty: a difference with previous information.
- *Uselessness*: when concluding results are useless for the environment (and the other agents).

We can observe the agent in using the knowledge it has about itself can locally judge the two first situations. The agent can analyse the third situation after a perception of the environment. This is the generic manner to define the engine of self-organization. For each level of the system, a set of NCS must be determined. This set must be as complete as possible. We instantiate it for the robot and the states agents in ***section 4.2 - Instantiation of the Model –***

3 A Multi-Level Architecture

In this section, we present our architecture to model a group of robots as Adaptive Multi-Agent Systems. First, we define the three levels that will be an adaptive multi-agent system: the robots, their inner states and their activity level. Later, we identify and describe the agents at each level and the non cooperative situations.

3.1 The Different Levels of the MAS

As a primordial motivation to easily model systems, the decomposition of a system in different levels of abstraction is a prevalent characteristic of our works. This decomposition enables to develop levels separately and to observe the phenomena that correspond to each level. At the robot level, the global modelled system is composed of several agents, the robots, which can be physically homogeneous or not. Each

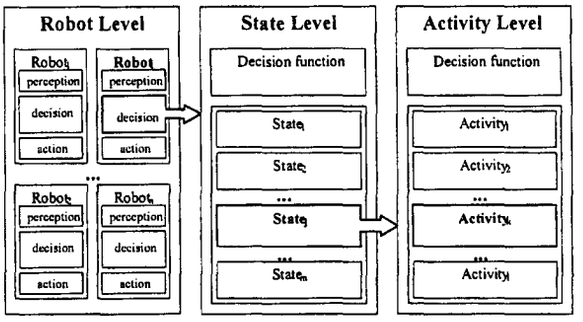


Figure 1. The three decomposition levels of the architecture.

robot is driven by its decision module that is composed of an adaptive multi-agent system too, where agents are states. At this level, named state level, the states must self-organize to give to the robot a coherent global behaviour.

This approach requires the definition and the identification of each agent in each level of the system. It is important to identify these levels because of the intrinsic multi-level characteristic of emergent phenomena. This appearance is a bottom-up phenomenon that can be propagated through the entire system and its level. Actually, an emergent behaviour can appear from the organization of the state agent, at the state level, to the robot level. Each robot is led by such behaviour so that an emergent global behaviour appears at the global level (or robot level). In our system we find three levels as in *figure 1*.

At the highest level, the *Robot Level*, the adaptive multi-agent system is composed of autonomous robots, which have to accomplish a collective task in a dynamical environment. For example, the collective's task of a collective may be resource transportation (Vaughan et al., 2000a) or a box-pushing task (Kube et al., 1996) (Matarić et al., 1995). The agents should be equipped as the robots that they represent. They have sensors, sonar for example, and actuators, as wheels, to be able to interact with their environment and to communicate indirectly with the other robots. They can also have communication equipment, as an IR-com port, to communicate directly.

At the mid-level there is the *State Level*. Each robot contains a multi-agent system in its Decision Module. This component that links the robot level and the state level is developed in ***section 2.2.4*** and ***section 5.2***. This module has to determine the behaviour the robot has to follow. The behaviour of a robot is directed by a sequence of inner states that are simple activities allowing the robot to accomplish its task. To obtain adaptive robots, a solution is to model robots with adaptive multi-agent components. Actually, the Decision Module has a representation of each state as agent. These agents must determine the right time to activate themselves to give to the robot a coherent behaviour. It is this self-organization that leads to the emergence of a robot's behaviour.

We can define the states as high-level primitives because they don't manipulate directly the actuators of the robot. For example, in the case of an ant having to bring resources to its nest, the states might be *exploration*, *exploitation*, *back to the nest* and *rest*. They might not be turn left, turn right, move forward, move backward or pick that manipulate directly the actuators (arms for example).

Other decompositions and level definitions are easily imaginable. As we said in the previous paragraph, states are high-level activities that don't manipulate directly the robot's actuators. So, what directly controls these actuators? The definition of another level is needed to complete this top-down definition of our architecture. This level is named *Activity Level*. Like robots, states are led in their local behaviour by a decision module that has to activate the right activity at the right time. A state needs different activities to be coherent. For example, an exploring robot has to know how to reach a resource when it detects one. More examples are given in ***section 3 - Example: a traffic Survey*** -. In fact, activities may manipulate directly actuators.

In this paper, we focus on the Robot Level and the State Level even if the Activity Level raised several questions which will be shortly developed.

3.2 High-Level Agents: the Robots

Robot agents have effect at the Robot Level. These agents are designed to control physical robots or to simulate them in an artificial life platform. They are composed of four distinct parts: *sensors*, *a decision module*, *actuators* and *a non cooperative situation detection module* (or *NCS detection module*). The three first components correspond to a classical principle of several works in Collective Robotics and Artificial Life. The last one, the NCS detection module is our contribution in the agent architecture. Agents are led by a classical three-phase life cycle:

1. The **perception phase** during which the robot updates its registers corresponding to its sensors and so updates their input vector composed of boolean values corresponding to the robot's point of view of its environment;
2. The **decision phase** during which the robot chooses an appropriate inner state in function of its input vector;
3. The **action phase** during which the robot updates its registers corresponding to its actuators (wheel speed, rotation angle...) in function of the taken decision of the previous phase.

The NCS module participates in the decision phase. If the robot locally assumes it is in a cooperative situation then it normally acts. Else, i.e. the robot locally infers it is in a non cooperative situation, it acts to come back to a cooperative situation.

As developed in the ***section 3.1 – The Multi-Agent System's Levels*** -, the robot can be composed of adaptive multi-agent systems as in others applications like where brokerage agents model their ontologies as multi-agent systems composed of words (Athanassiou et al., 1999).

Sensors represent the link between the robot and its environment. They allow the robot to construct a partial representation of the world where it is plunged. For the most part of them, their ranges are narrow. This is why such robots are suitable for autonomous agents that obey to the *locality principle*: an agent has a local perception of its environment. This principle is a "sine qua non" condition for an agent to be, as says Ferber (1995). Sensors update the robot's input vector.

NCS detection module is coupled with the sensors and taking place during the perception phase, this module analyses the input vector. In function of its inputs, including what a robot knows about itself - its current state for example -, the module determines if the robot is in a non cooperative situation. In such a situation, the NCS detection module has to send a message to the decision module to treat this particular situation.

Actuators assume the link between the robot and its environment. Thanks to actuators, the robot can move (legs or wheels) or can pick some objects, for example, that modifies the environment. Registers containing values representing their position, orientation or whatever lead actuators. The only representation the robot has of its actuators is this set of values.

The *decision module* selects an inner state in function of the input vector. In our architecture, the decision module uses another multi-agent systems composed of state agents, taking place at the State Level. The problem of the choice of the right state at the appointed time appears in several works in Robotics (Matarić, 1994). For our part, we explore a new way for the decision, the decision by adaptive multi-agent system and a simple reactive decision.

- **Reactive decision**: at each possible input vector, the module associates a state. This kind of decision is very efficient but is not flexible. It needs a complete exploration of the input vector space at the conception of the robot even if factorisations may be possible.
- **Agent-based decision**: during the decision phase, each state agent evaluates its wish to act at this time. A state agent has beliefs on input-vector and affects weight to each of them. The most pertinent state will be activated, i.e. each state agent sends a value corresponding to its wish to be activated and a decision function chooses the best (the agent with the highest value for example). With such a decision module, robots are able to learn from state-agents' self-organization as explained in ***section 5.1 - Learning from Self-Organization*** -.

The question of the decision can be raised at the state level too. Therefore, a state agent must activate the right activity at the proper time to be coherent. It is the motivation of the decomposition of the agents in multi-agent systems.

3.3 Mid-Level Agents: the States

The state agents appear in the decision module. A state represents a behaviour a robot can have. The role of the state agents is to activate themselves at the proper time to control coherently the robot.

A state agent has two tasks to accomplish. Firstly, it has to select the right activity at the proper time. Secondly, it must send a value to the decision module.

In fact, the global behaviour of a robot, i.e. a sequence of activities, can be represented by a transition graph as in the methodology CIRTA of Labbani-Igbida et al. (1998). In this graph, two levels may appear: the state level and the activity level. The goal is to find the graph robots have to follow. Keeping in mind we work with the emergent concept, this graph does not have to appear at the robot level. The robot should have to construct it by learning.

3.4 Related Works

Our architecture is similar to the behaviour-based architecture of Brooks (1986) or Matarić (1994). In fact, our states and activities correspond to their *behaviours*. In these architectures, each agent has a set of simple behaviours, which enable the agent to accomplish its task.

The choice between the different behaviours at a given time can be done by different procedures: arbitration, subsumption, etc. In our works states are agents having to choose by themselves the right state at the proper time. The choice is not a centralized procedure but it is distributed between the state agents.

4 TRAFFIC SURVEY

To explain the presented architecture, we develop an application. This study shows a common problem in Collective Robotics: spatial interference. The result of this application is the observation of a global emergent behaviour. We show that a stream of a collective through corridors is a global emergent behaviour.

4.1 Presentation of the Problem

The resource transportation problem is a classical task in Collective Robotics due to its inspiration from insect communities (Kube et al., 1992) (Vaughan et al., 2000a). The task of the robots is to transport resources from a claim area to a laying area. These areas are

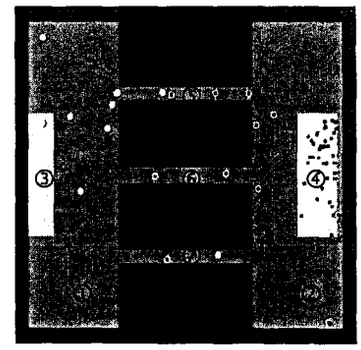


Figure 2. An example of environment for the stream emergent behavior.

situated in different rooms separated by narrow corridors. A spatial interference problem appears as in the survey of Vaughan et al. (2000b). Once a robot is engaged in a corridor, what has it to do if another robot comes in the opposite direction? In this survey, they use an aggressive competition to resolve the problem. For our part, we use the NCS concept.

The configuration of the environment has of course a great importance on the appearance of emergent phenomena. As *figure two* shows, each room (1 or 2) contains an area (3 or 4 respectively) at the opposite of the other room to force robots to enter in a room to realize their activities. Corridors (5, 6 and 7) are narrow (more than the size of a robot but less than the size of two robots) and their length can be parameterised. In fact, the length has an importance on the appearance of a stream direction. Either the corridor is shorter than perception range of the robot, or the corridor is longer. In the first case, a robot can see another robot engaged in a corridor before coming in. In the second case, a robot cannot know if another is engaged and has to come in without certitude. The number of corridors may have an importance too. In this article, we only present the case with two long corridors.

4.2 Instantiation of the Model

Now, we present how to use our architecture for the resource transportation task.

4.2.1 The Robots

To complete the task, a robot must have some sensors and actuators: *short-ranged sensors* to avoid obstacles, an *object sensor* to distinguish robots from resources and an *area sensor* to detect areas and corridors (each area or corridor can have a proper colour); *two wheels* to move in any direction and a pick-up unit (such a clamp) to pick resources. The sensors enable a robot to construct its input vector.

We have determined a set of inputs such as `seeResource`, `inClaim` or `inCorridorNumber`. These inputs are booleans or list of Booleans according to *table 1*. This set of inputs is called the input vector.

4.2.2 States and Activities

We have determined the following states and activities:

- **The Claim State**: the state a robot must have when he have to take a resource. This state uses the following activities:
  - *Seek Resource*: the robot is exploring the rooms to find a resource;
  - *Reach Resource*: the robot is moving to a resource (it's able to avoid obstacles too);
  - *Pick Resource*: the robot is picking a resource which is close ranged;
- **The Laying State**: the state a robot must have when he have to drop a resource. This state uses the following activities:
  - *Seek Area:* the robot is exploring the rooms to find the Laying Area;
  - *Reach Area*: the robot is moving to the Laying Area resource (it's able to avoid obstacles too);
  - *Drop Resource*: the robot is dropping a resource;
- **The Travel State X**: the state a robot must have when he have to cross the corridor X. This state uses the following activities:
  - *Seek Corridor X*: the robot is exploring the rooms to find the corridor X;
  - *Reach Resource X*: the robot is moving to the corridor X (it's able to avoid obstacles too);
  - *Pick Resource X*: the robot is crossing the corridor X;

Each state uses a set of activities that can be summed up as *seek*, *reach* and *act*. The robots should have to: seek a resource, find it, reach it, pick it, then seek a corridor, find it, reach it, cross it and then seek the laying area, find it, reach and drop the carried resource. This behaviour corresponds to a transition graph where conditions may be very complex because of the number of parameters in the input vector. A solution is to group activities in states, as in *figure 3*. Conditions are factorised and easier to define.

4.2.3 NCS rules

We shall define the NCS corresponding to the robot level. The NCS corresponding to the state level will not be explain because will only focus on the appearance of emergent behaviour at the robot level.

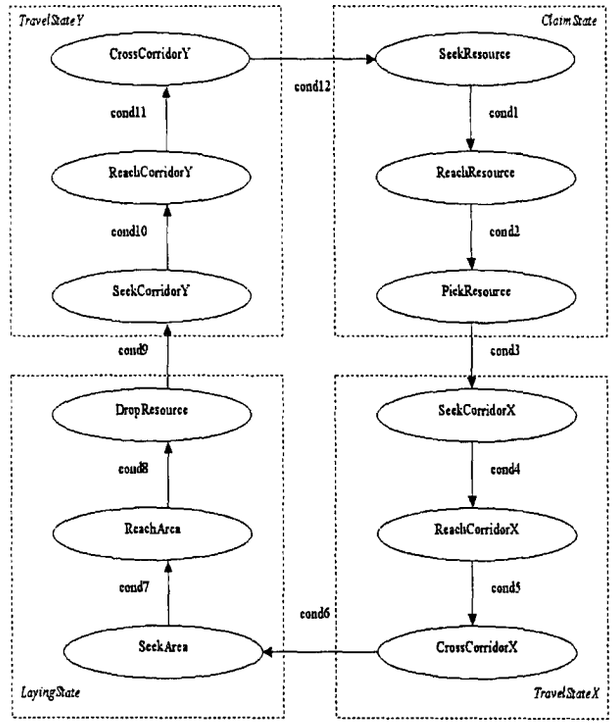


Figure 3. The State Level and the Activity Level appear in the transition graph.

1. A first robot A, which is not carrying a resource, is reaching a corridor and sees another robot B, which is carrying a resource. The robot A is reaching a corridor which is frequented by robots carrying resources and it may disturb them;
2. A first robot A, which is not carrying a resource, is reaching a corridor and sees another robot B which is immobile and which is not carrying a resource. The first robot A is reaching a corridor which is blocked for carrying robots;
3. A first robot A, which is carrying a resource, is reaching a corridor and sees another robot B, which is not carrying a resource. The robot A is reaching a corridor which is frequented by carrying robots and may disturb them;
4. A first robot A, which is carrying a resource, is reaching a corridor and sees another robot B which is immobile and which is carrying a resource. The robot A is reaching a corridor which is blocked for robots which are not carrying resources;
5. A first robot A, which is not carrying a resource is reaching a corridor and sees another robot B carrying a resource. A and B are crossing a corridor which is frequented by robots which do antinomic jobs;

6. A first robot A, which is carrying a resource is crossing a corridor and sees another robot B, which is not carrying a resource. Same as (5);
7. A first robot A is crossing a corridor and sees an immobile robot. The corridor is blocked. It may be due to robots in NCS (5) or (6);

4.3 Preliminary Experiments

At this stage, the system is tempting but limited. We shall expose in this section first results, which have motivated our wish to extend our system, notably by learning capacities. Experiments have been realized on the *oRis* platform developed at the *ENIB* (Harrouet, 2000). In this section, we will show different results from measurements on the collective for different configuration of agents and we will conclude on the need to equip our agents of learning capabilities.

4.3.1 Robots without NCS Detection Module

Results of *figure 5* show the corridor frequenting. The configuration of experimentation is two corridors and twenty robots that are unable to detect NCS. Two curves represent the number of robot crossing a corridor. Each curve corresponds to a direction: from the claim room to the laying room, or from the laying room to the claim room. As an observation, the corridors are not dedicated to a direction. There is no emergent behaviour.

4.3.2 Robots with NCS Detection Module

Results of *figure 6* have been obtained with the same configuration except that the robots can detect and treat NCS. However, there is no emergent global behaviour.

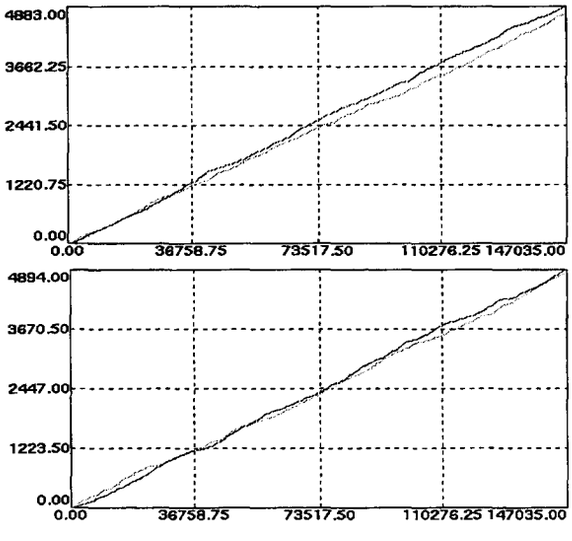


Figure 6. Number of incoming robots in the corridors (first corridor at top and second corridor at bottom) in function of simulation's step time, for robots with NCS detection module.

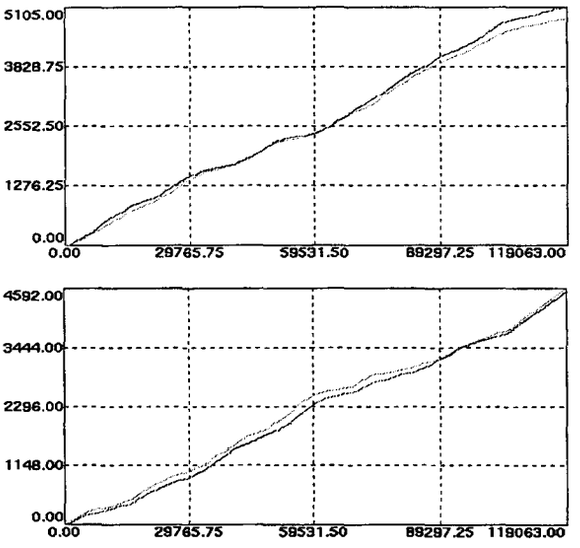


Figure 5. Number of incoming robots in the corridors (first corridor at top and second corridor at bottom) in function of simulation's step time, for robots without NCS detection module.

4.3.3 NCS are not Totally Exploited

To be able to detect and resolve immediately NCS seems not to be sufficient to observe emergent behaviour. The previous results underline the need to add learning capacities to robots. In fact, conflicts are not due to the current state of a robot but to a previous one, cause to the length of the corridors.

5 Learning and Decision

In the previous section, we have shown the need to endow our robots of learning abilities. We develop now the concept of learning by self-organization, which is the way for our system to adapt itself to its environment.

5.1 Learning from Self-Organization

Learning for a system S consists in modifying autonomously its function f_S to adapt itself to its environment. In this case, the environment is a given constraint. Each agent A_i of a system S achieves a partial function f_{Ai} of the global function f_S. f_S is the result of the combination of the partial functions f_{Ai}, noted by the operator "$\circ$". The combination being determined by the current organization of the parts, we can deduce:

$$f_S = f_{A1} \circ f_{A2} \circ \ldots \circ f_{An}$$

As generally $f_{A1} \circ f_{A2} \neq f_{A2} \circ f_{A1}$, by transforming the organization, you change the combination of the partial functions and therefore you modify the global function f_S. Therefore, this is a way to adapt the system to the environment.

The theorem we carried can be expressed like this:

Theorem 2. *Any system having a cooperative internal medium is functionally adequate.*

Each agent has to be in cooperative interaction with the others so as the totality be in cooperative interaction. It means that each agent who locally detects non cooperative situations must try to change the organization in order to get in a new cooperative state. It might be restrictive to use the principle of self-organization, that is to say the research of an optimum organization, as only learning mechanism. However, structuring a system in levels of different granularity allows a learning not only at the level of the organization but, also, at the level of skills. Indeed, the skill of an agent changes if its internal organization (the agents it is made of) is changed. So, the level's organization is chained to the one of the upper level. In fact, each agent could be also a multi-agent composed of cooperative sub-agents at a lower level.

5.2 Learning at the Robot Level

Learning at the Robot Level can be effectuated by an agent-based decision module. A change in the organization of the state agents corresponds to a change of the robot agent skill. When a robot detects a NCS, it has to learn in order to come back to a cooperative situation.

The environmental feedback is *implicit*[2], i.e. this is the robot who detects this feedback. A robot agent learns on its skill: it changes its graph of states in order to change its global behaviour. To change its graph, it must change the organization of its state agent by sending a message to its decision module when it detects a non cooperative situation.

5.3 Learning at the State Level

At the State Level, in the decision module, each state agent evaluates its wish to be activated. For the collective of state it corresponds to an organization. States must learn when a wrong state is activated at an improper time. States detects such a situation when the robot agent sends a NCS message to its decision module.

To evaluate its will to be activated, a state agent must calculate a wish value. To calculate this one, the state agent multiplies a modified input vector (e_i) with weight vector (c_j) corresponding to the state agent's belief on a component of the vector.

$$[e_1 \quad e_2 \quad \dots \quad e_{n-1} \quad e_n] \bullet \begin{bmatrix} c_1 \\ c_2 \\ \dots \\ c_{n-1} \\ c_n \end{bmatrix} = \textit{value to return}$$

The modified inputs e_i are equal to *1* for an input vector value of *1*, and equal to *-1* for an input vector value of *0*. The e_i values correspond to "is the input important for the decision?" and the c_j values correspond to "how much important the input value is". Learning will be done by modifying the weights c_j when the decision module receives a NCS message from the robot (the weight decreases) or when the state corresponding to the state agent is chosen (the weight increases). After the calculus of their values, the agents compare them and decide which state will be activated.

This learning model is close to the Reinforcement Learning model or the model proposed by Matarić (1994).

6 Results

In the previous sections, we have seen the necessity to supply our robot with learning abilities to reach our goal: emergence of the stream global behaviour. Therefore, we have developed an agent-based decision module. In this section, we present results showing how a collective of robots, which are supplied with such a module, evolve by self-organization. As in ***section 4.3 – Preliminary Experiments –***, the collective is composed of 20 robots plunged in a two-corridor environment. Finally, we conclude on the appearance of coherent group behaviour.

6.1 Robots with Agent-Based Decision Module

In the *figure 7*, the lighter curve corresponds to robots which not carrying resources and the darker one corresponds to the robots carrying resources. Results show that the two "classes" of robots that cross the corridors (*carrying robots* and *not carrying robots*) are well dissociated. Carrying robots appropriate the second corridor. This phenomenon is due to learning ability: it does not appear with a collective of robots without decision module. In fact, at the beginning of the simulation, robots experiment numerous NCS because they disturb themselves at the entries of the corridors. Bit by bit, they change their transition graph.

The dissociation between corridors does not correspond to a cast formation as referred by Balch (1997), because this is a dynamic phenomenon where robots constantly move from class to class. Therefore, all the robots follow the same circle: claim room, second corridor, laying room, first corridor, and then claim room ... As the environment includes only two corri-

[2] In opposition to explicit feedback, where an external omniscient entity decide for each agent if its action is good or not.

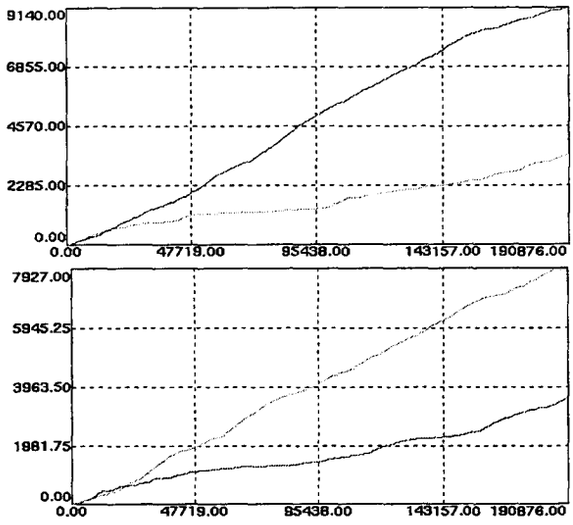


Figure 7. Number of incoming robots in the corridors (first corridor at top and second corridor at bottom) in function of simulation's step time, for robots with agent-based decision module.

dors, it seems to be evident. However, what if it includes more then two corridors? In this case it seems to be interesting to observe cast formation.

The results show that the two corridors are frequented by the two "classes" of robots. This is a consequence of the used method to count the robots. To obtain these results, these are the incoming robots that are counted, not the robots that completely cross the corridors. Therefore, robots that unfortunately come in the corridor to avoid an obstacle are counted.

We can see that this dissociation is done in a reduce time, at the beginning of the simulation (20,000 steps). It shows the efficiency of the learning by self-organization for a wide solution space.

6.2 Emergence of a Global Behaviour

Robots self-organize to transport resources by crossing specific corridors. Nevertheless, we have not coded in the robots an algorithm to. The micro-level specification leads to the appearance of a coherent emergent phenomenon: the stream, i.e. the dedication of corridors to specific direction.

7 Conclusion and Perspectives

In this paper, we develop an example of a Collective Robotics problem: the resource transportation through corridors. This problem is tackled in the idea to observe emergent phenomenon from the collective. After the presentation of our architecture and the study of the task, we present some results obtained by simulation on the agent-oriented platform *oRis*. Robots have to be equipped with learning abilities to observe the emergent stream. This conclusion is a motivation to develop an agent-based decision module. The expected result appears: the stream behaviour emerges from the collective.

The developed system is incomplete, but offers several perspectives for the future:

- **Optimisation of the NCS resolution:** because they lead to short-time blockages at the entries of the corridors;
- **Experimentation of a "Logical Adaptive Network":** where the decision is done by an adaptive multi-agent system composed of "nand" agents which emulate the transition graph;
- **Other behaviours and cast dynamic study**: such as "box-pushing" with team or hierarchy;
- **Learning by cooperation**: the robots are able to communicate and to share their experiences.

The results obtain with our present system are encouraging for two reasons. First, the method that is used to specify and design the system is confirmed to be efficient for the study of complex system with emergent functionality. Finally, the AMAS theory seems to be pertinent in Collective Robotics domains, even if our application is only a laboratory case.

In a more general way, our works focus now on a methodology, which is based on the AMAS theory and the UML notations, to design adaptive multi-agent systems with emergent functionalities. This methodology, named ADELFE, regroups several partners[3] to develop a toolkit for engineers and will be implemented in the OpenTool application, which is provided by TNI firm.

References

S. M. Ali and R. M. Zimmer. The question concerning emergence. In *Computing Anticipatory Systems: CASYS - First International Conference, D.M. AIP Conference Proceedings* 437, pp 138-156, 1998.

Athanassiou, Chirichescu, Camps, Gleizes, Glize, Lakoumentas, Léger, Moreno, Schlenker. Abrose: A Co-operative Multi-Agent Based Framework for Marketplace - *IATA, Stockholm*, Sweden August 1999

T. Balch. Social Entropy : a New Metric for Learning Multi-Robot Teams. In *Proceedings, 10th International FLAIRS Conference (FLAIRS-97)*. 1997

C. Bernon, V. Camps, M.P. Gleizes, P. Glize. La conception de systèmes multi-agents adaptatifs : contraintes et spécificités. In *Atelier de Méthodologie et Environnements pour les Systèmes Multi-Agents*

[3] RNTL Project Partners: ARTAL Technologies, IRIT, L3I and TNI.

(SMA 2001), Plate-forme AFIA, Grenoble, June 2001.

R. Brooks. A robust layered control system for a mobile robot. In *IEEE Journal of Robotics and Automation*, volume 2, pages 14-23, 1986.

J. Ferber. *Les systèmes multi-agents : Vers une intelligence collective*. InterEditions, Paris, 1995.

M.-P. Gleizes, V. Camps, and P. Glize. A theory of emergent computation based on cooperative self-organization for adaptive artificial systems. In *Fourth European Congress of System Science, Valencia, Spain*. 1999.

J. Goldstein. Emergence as a construct: History and issues. In *Emergence : a Journal of Complexity Issues in Organizations and Manadgement*. The New England Complex Systems Institute, 1(1):49-72, 1999.

F. Harrouet. oRis : s'immerger par le langage pour le prototypage d'univers virtuels base d'entités autonomes. PhD thesis, Université de Bretagne Occidentale, 2000.

J. Holland. *Emergence: From Order to Chaos*. Oxford University Press, Oxford, 1998.

C. R. Kube and H. Zhang. Collective robotics: from social insects to robots. *Adaptive Behaviour, 1994*, 2(2):189-218.

C. R. Kube and H. Zhang. The use of perceptual cues in multi-robot boxpushing. In *1996 IEEE International Conference on Robotics and Automation*, pages 2085-2090, 1996.

O. Labbani-Igbida, J.-P. Müller, and A. Bourjault. Cirta: An emergentist methodology to design and evaluate collective behaviours in robots' colonies.In *Proceedings of the 1st International Workshop on Collective Robotics (CWR-98), volume 1456 of LNAI*, pp 72-84, Berlin, 1998.

MARCIA Group. Auto-organizsation := 'évolution de structures? In *Journées du PRC GDR Intelligence Artificielle: les systèmes multi-agents, Toulouse*. 1996.

M. J. Matarić. *Interaction and Intelligent Behavior*. PhD thesis, MIT, May 1994.

M. J. Matarić, C. Nilsson, and K. Simsarian. Cooperative multi-robot box-pushing. In *IEEE International Conference on Intelligent Robots and Systems, volume 3*, pages 556-561, 1995.

T. Sontheimer, P. Cornuau, J.J. Vidal, P. Glize. Application d'un système adaptatif pour la prévision de crues dans le bassin de la Garonne – Un Modèle Emergent. In *Conférence SIRNAT'01*, 2001.

R. Vaughan, K. Støy, G. Sukhatme, and M. Matarić. Blazing a trail: Insect-inspired resource transportation by a robotic team. In *Proceedings, 5th International Symposium on Distributed Robotic Systems*, October 2000.

R. Vaughan, K. Støy, G. Sukhatme, and M. Matarić. Go ahead make my day: Robot conflict resolution by aggressive competition. In *Proceedings, 6th International Conference on Simulation of Adaptive Behaviour*, pp 491-500, 2000.

Environmental Risk, Cooperation and Communication Complexity

Peter Andras, Gilbert Roberts, John Lazarus
Department of Psychology
University of Newcastle
Newcastle upon Tyne, NE1 7RU, UK
{peter.andras;gilbert.roberts;j.lazarus}@ncl.ac.uk

Abstract

The evolution of cooperation and communication in communities of individuals is a puzzling problem for a wide range of scientific disciplines, ranging from evolutionary theory to the theory and application of multi-agent systems. A key issue is to understand the factors that affect collaboration and communication evolution. To address this problem, here we choose the environmental risk as a compact descriptor of the environment in a model world of simple agents. We analyse the evolution of cooperation and communication as a function of the environmental risk. Our findings show that collaboration is more likely to rise to high levels within the agent society in a world characterised by high risk than in one characterised by low risk. With respect to the evolution of communication, we found that communities of agents with high levels of collaboration are more likely to use less complex communication than those which show lower levels of collaboration. Our results have important implications for understanding the evolution of both cooperation and communication, and the interrelationships between them.

1 Introduction

Understanding how cooperation between unrelated individuals can arise in animal and human societies has puzzled evolutionary theorists. Early solutions to the problem were found in terms of reciprocal altruism; the mutual exchange of benefits between pairs of individuals (Trivers 1971, Axelrod & Hamilton 1981, Axelrod 1984, Roberts & Sherratt 1998). More recently, 'indirect reciprocity', in which individuals who are seen to be more generous receive more help from others, has been proposed as an additional route to cooperation (Alexander 1987, Nowak & Sigmund 1998, Leimar & Hammerstein 2001).

In these models individuals have information about the past behaviour of others on which to base decisions but there is no communication of intentions: individuals simply act; cooperating, defecting or declining to interact. (This is in contrast to the evolutionary modelling of competitive behaviour in which signalling has played a central role: e.g. Maynard Smith 1974, Enquist 1985.) Although there seems to be little theoretical work on intentional signalling in the context of cooperation, arbitrary signals correlating with altruism (the 'green beard effect', Dawkins 1976), tags indicating individual identity (Riolo et al. 2001) and signalling of partner quality (Leimar 1997) have been considered. While existing evolutionary models of cooperation are important in examining the minimal conditions for the evolution of cooperation they are also impoverished - at least for the human case - in excluding the possibility that, intentionally or unintentionally, individuals may communicate their intention to cooperate before interacting. In the development and maintenance of human relationships cooperation is accompanied by signals of short-term intentions and longer term commitment. Honest communication, and consequent trust, are of the greatest importance for the development of a stable collaborative relationship; deceit and mistrust are inimical to it (Boon & Holmes 1991).

In this paper we develop an agent world model to examine the evolution of cooperation when individuals communicate their intentions. This communication helps individuals to decide whether to enter into cooperation with another: it allows partner choice. Many of the earlier models provide no such choice, so that cheats can be avoided only if individuals have information on their past behaviour. When the behaviour of others is to some extent predictable, however, individuals can derive the intentions of others and cooperators can choose to interact with other cooperators while cheats can be ostracized (Roberts 1998).

We ask how cooperation and communication respond to variation in the risk or complexity of the environment. Risk and complexity are important general properties of the environment that impact on an individual's success in ways that can be influenced by cooperating with others. For example, resources may be predictable (low risk) or unpredictable (high risk), and threats from predators or competitors may vary in a similar way. Resource acquisition, the avoidance of predators and success in competition can all be enhanced by collaboration with others.

The second focus of the paper is how the complexity of communication itself evolves in this context and whether it differs between cooperators, cheaters and those who decline to interact at all. This interest in the communication of intentions follows other work in artificial societies (e.g. Schillo et al. 2000), which investigate how collecting information about the intentions of other agents can enhance the development of collaboration.

Our agents communicate by sequences of signals, each of which informs the potential partner about the cooperative intentions of the agent. The reliability of the signals differs between agents, who use the information from communication in present and past interactions to make decisions about whether to share resources with a potential partner. The interaction between agents occurs within a risky environment where risk refers to the variability of the gains that result from cooperative behavior. Our results will be applicable to agent societies, animal societies, and human social systems.

The rest of the paper is structured as follows. First, we describe the agent world model. Next, we present the simulation results. Finally, we discuss the implications of our results.

2 The agent world

In this section we describe the world of our agents. We start with the description of the environment, followed by the description of the agents, communication processes, resource management, the offspring generation rules, and we close with the description of the evolution of the agent society. We also present some argumentation to support our choices with respect to the implemented principles, rules and methods.

2.1 The environment

The environment of our agent world is characterized by a given risk. The environmental risk represents in a compact way the complexity of the surrounding environment. The environmental risk is implemented as the variance of the agent's resource regeneration process.

2.2 The agents

Our agents dispose over some generic resources that they use to maintain themselves and to reproduce.

Each agent speaks the same communication language consisting of the symbols: '0','s','i','y','n','h' and 't'. The meaning of the communication symbols are as follows: '0' – no intention of communication, 's' – start of communication, 'i' – maintaining the communication, 'y' – indication of the willingness to engage into resource sharing, 'n' – indication of no further interest in communication, 'h' – effective sharing of the resources, 't' – not sharing the resources after an indication of willingness to engage into sharing. The last two symbols, 'h' and 't' actually mean the resource-sharing or no-resource-sharing actions. The first four symbols are ranked according to their positive contribution towards engagement in sharing (the least positive is the '0' and the most positive is 'y').

The agents generate communication units (i.e., one of the above symbols) when they engage in communication with another agent. Each agent has its own realization of the language. This language is represented in the form of a two-input probabilistic automaton (i.e., it is equivalent of a probabilistic push-down automaton). The language units are production rules of the form

$$\begin{aligned} L:\ & U_{current}, U'_{current} \rightarrow_{p1} U_{new,1} \\ & U_{current}, U'_{current} \rightarrow_{p2} U_{new,2} \\ & \cdots \\ & U_{current}, U'_{current} \rightarrow_{pk} U_{new,k} \end{aligned}$$

where $U_{current}$ is the last communication unit produced by the agent, $U_{current}'$ is last communication unit produced by the partner of the agent, $U_{new,1}$, $U_{new,2}$,... $U_{new,k}$ are the new communication units that can be produced by the agent, and p1,p2,...,pk are the probabilities of production of these communication units, p1+p2+...+pk=1 (an example of a such rule is: L: i,i' $\rightarrow_{0.4}$ y, i,i' $\rightarrow_{0.5}$ i, i,i' $\rightarrow_{0.1}$ n that means that after producing the symbol 'i', and receiving a symbol 'i' from the communication partner, the agent will produce the symbol 'y' with probability 0.4, the symbol 'i' with probability 0.5, and the symbol 'n' with probability 0.1).

The language units obey intention consistency rules, i.e., if U0,U1,U2, and U3 are communication units, and U2 is equally or more positive than U1, and U3 is equally or more positive than U2, and L1 is a language unit that produces U3 after U1 and receiving U0 , and L2 is a language unit that produces U3 after U2 and receiving U0, then the probability of producing U3 using L2 is equal or higher than the probability of producing U3 using L1. Similarly, if L1 is a language unit that produces U3 after U0 and receiving U1 , and L2 is a language unit that produces U3 after U0 and receiving U2, then the probability of producing U3 using L2 is equal or higher than the probability of producing U3 using L1. In other words the intention consistency rules mean that more positive inputs are more likely to lead to positive outputs than are less positive inputs. This choice of the intention consistency rules is in agreement with human and animal behavior, where the expression of friendly signals is more likely to be followed by further friendly signals by the same individual than non-friendly signals.

Each agent has a characteristic intention, which indicates the extent to which it is willing to share resources with other agents. This sharing intention determines the probability of the y,y' → h production rule.

The agents are equipped with a memory. The memory of the agents can store the experiences of collaboration with the last M different partners (M=10 in our implementation). The memory of the agents also fades with time, and if they don't meet an old partner for long time they forget their memories about this partner. For each memorized partner the agent keeps the score of the successful and unsuccessful meeting (i.e., successful means a meeting that led to getting shared resources from the partner).

The agents are located on a two dimensional plane, and they may change their location. The location of an agent determines the neighbourhood of the agent that consists of the N (N=10) closest agents.

The agents live for T (T=60) time units. In each time unit they try to find a collaboration partner in their neighbourhood. At the end of their life time the agents produce their offspring.

2.3 Communication processes

After selecting a collaborating partner the agents may engage in a communication process. The communication process starts properly after both agents communicated the 's' symbol. We set a limit (L_1) for the preliminary communication. If the two agents do not reach the proper start of the communication in a communication of length L_1 we consider that they stop their communication at this moment.

During the communication process the agents use their own realization of the common language to produce communication units. The communication process ends either with the communication of an 'n' symbol (i.e., signalling no further interest), or with the communication of the 'y' symbol by both partners. After this each agent decides whether to share or not to share their resources with the other agent by producing the action symbol 'h' or 't'. We impose a communication length limit (L_2) on the proper collaboration oriented communication. If the agents do not reach the stage of communicating the 'y' symbols in L_2 communication steps, we consider that they stop their communication.

At the start of each communication process, the agents update their language unit probabilities according to their memories of the agent they are currently interacting with (note that if there had been no previous interaction with this agent there is no update). The updated version of their language applies only to the present communication process. In the case of more positive experiences (those that led to sharing) the probabilities leading to more positive symbols are increased, while in the case of negative experiences these are decreased. The probabilities for each language unit are normalized after effecting the above changes (i.e., the probability of producing all the allowed new symbols is always one for each language unit).

During each communication process, as an agent produces equally or more positive symbols their willingness to share increases. We note that although this increase happens in all agents, those who have very low intention to collaborate will increase an originally low probability, which means that they will not necessarily share at the end of the communication process. We adopted this collaboration willingness increase principle in conformity with human and animal behavior, where a sequence of expression of friendly signals increases the likelihood of the friendly ending of the interaction, even if the original intentions were less friendly.

2.4 Resource management

The agent dispose over their own resources that they use to maintain themselves and reproduce. In each turn the agents use their available resources to produce new resources. If they manage to find a partner who is willing to share its resources they can use the combined resources to generate the new resources for the next turn.

The mean resource generating function is a squashing function of the form:

$$\overline{R}_{new} = a \cdot \frac{1}{1 + e^{-R+R_0}}$$

where R is the amount of available resources, and R_0 and a are parameters. Operating at the convex half of the squashing function (i.e., $R < R_0$) means that using more added resources is more advantageous than using the resources separately.

Resource generation happens in a probabilistic manner. The environmental risk specifies the variance of the resource regeneration process. The amount of new resources is found by taking a sample from a uniform distribution that has the calculated mean and the variance specified by the environmental risk. We use the notation N(R) for the amount of new resources generated by using R amount of available resources.

The variance of the resource regeneration increases with the time spent in negotiation about resource sharing. This risk increase principle is in agreement with how environmental risk changes in the real world. To exemplify it we consider an example from business. If two companies start negotiations about a joint business, lengthy negotiations may proceed while a

competitor enters in the market, and the final gain of the two collaborating companies will be reduced. At the same time lengthy negotiations may lead to a well-designed contract that makes possible to avoid future impasses, making the collaboration more profitable. If the negotiations are short, the deal is made quickly, and the companies may start gaining some new market share. At the same time they may run into some unregulated disputes that may slow down their cooperation and the increase of their market share. As we can see from this example, if the communication process is short the variance of the expected benefits is likely to be smaller than in the case when the communication process becomes lengthy.

When two agents meet, having resources R1 and R2, and they both decide to share their resources they may receive extra new resources. The extra resources for both partners are calculated as the half of the difference

$$N(R1+R2) - (N(R1) + N(R2))$$

Such agents are called collaborators.

If an agent is engaged in a communication process, convinces its partner to share, but then withholds its own resources from sharing, it is called a cheater. The gain of a cheater is the whole amount of the difference

$$N(R1+R2) - (N(R1) + N(R2))$$

In such case the one that is cheated generates only N(R2) new resources for itself, i.e., it does not benefit from the sharing.

If two agents select each other as communicating partners, but they do not manage to decide about the sharing of their resources (i.e., their communication end with an 'n' symbol) we call them non-collaborators.

If an agent does not reproduce enough resources to maintain itself (i.e., the maintenance costs are higher than the amount of own resources) the agent reaches the zero resource level and dies.

2.5 Offspring generation

When the agents reach the end of their lifetime they generate their offspring. The number of the offspring depends on the available resources of the agent.

If the agent has R resources, and the mean amount of the resources in the agent society at that moment is R_m, and the standard deviation of the resources is R_s, then the number of offspring of the agent is calculated as

$$n = \alpha \cdot \frac{R - (R_m - \beta \cdot R_s)}{R_s} + n_0$$

where α, β, n_0 are parameters.

If n is negative or R=0 we consider that the agent produces no offspring. If $n > n_{max}$, where n_{max} is the allowed upper limit of offspring, we cut back n to n_{max}.

In order to avoid strong generational effects the newly generated offspring have random ages between 1 and A_0.

The offspring of an agent inherit from their parent its language and collaboration intention with small random modifications. They also inherit the resources of their parent equally divided between the offspring.

2.6 The evolution of the agent society

At the beginning we start with randomly initialised agents, i.e., the transition probabilities of their language units, their collaboration intentions, initial resources and initial positions are set randomly.

The agent society evolves through the interaction and reproduction of the agents. The agents search for collaborating partners. They try to share their resources, or to cheat the collaborating partner, or they may not manage to make the decision about sharing. In each turn each agent may choose one partner from its neighbours. After each turn the agents make a random move, changing their position, and possibly finding a new neighbourhood.

The agents regenerate their resources alone or in collaboration with another agent in each turn, and they pay a part of their resources to maintain themselves. At the end of their lifetime the agents generate their offspring if they have enough resources.

3 Simulation results

This section presents our simulation results. The objective of these simulations were twofold. First, to determine how environmental risk affects both the level of collaboration, and the complexity of communication. Second, to examine how the various strategists (collaborators, cheaters, non-collaborators) differ in complexity of their communications in an evolving society. The results are presented in this order, after examining some general effects of risk level on the agent society.

We selected five levels of risk in the range of 0.1 – 0.9 (the risk levels were 0.1, 0.2, 0.5, 0.7, and 0.9). We measured communication complexity by measuring the average length of the communication processes.

For each risk level we run 20 simulations to obtain valid estimates of average values and variances of the measured variables. The number of agents in each simulation was the same at the beginning (1500). We run each simulation for 400 time units, or until the

agent population died out or reached the maximum allowed level of number of individuals (5000).

3.1 General effects of the environmental risk

To see the general effects of environmental risk on the agent society we looked at how the number of agents and the resource level varied with time. We considered the average amount of resources separately for collaborators, cheaters and non-collaborators.

Figure 1 shows the average number of agents in the agent societies for the five risk levels. Figures 2 – 4 show the change over time of the average amount of resources of collaborators, cheaters and non-collaborators.

The first segment of dropping in the graphs represents the period when the randomly initialised population selects those who are able to survive. This segment corresponds to one generation (i.e., around 60 time units).

Following the initial drop the societies start to grow in number and in average amount of resources in all cases. The graphs show that this growth happens much faster in agent societies living in low risk environment than in those which live in high risk environments.

These results confirm the standard expectation that the average level of populations and their available resources is lower in high risk environments than in low risk environments.

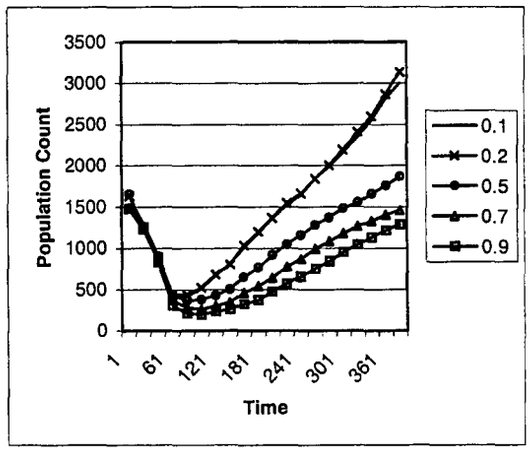


Figure 1: The evolution of the number of agents in agent societies living in environments with different risk levels.

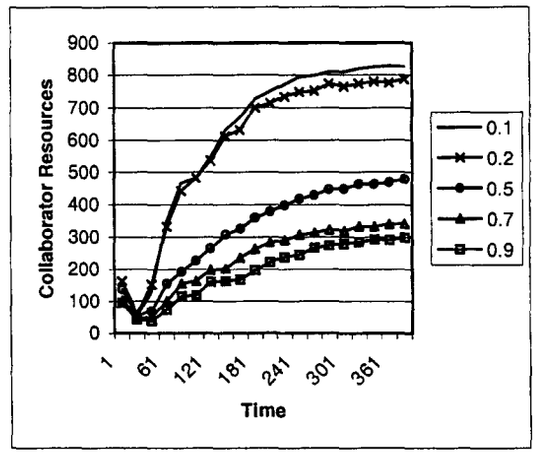


Figure 2: The evolution of the average amount of resources of collaborators in agent societies living in environments with different risk levels.

3.2 Environmental risk and the level of collaboration

To analyse the effect of the environmental risk on the level of collaboration we looked at the percentage of collaborators, cheaters and non-collaborators within the society (note that the percentage of those who were cheated is the same as the percentage of cheaters). The change of these percentages over time is shown in Figures 5 – 7.

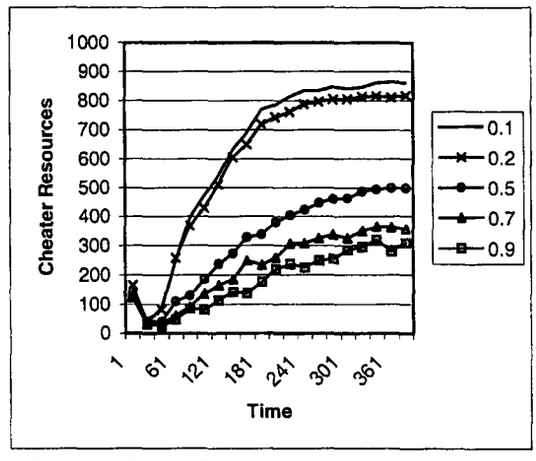


Figure 3: The evolution of the average amount of resources of cheaters in agent societies living in environments with different risk levels.

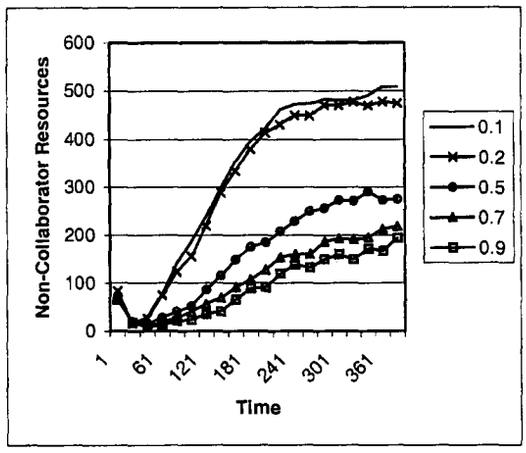


Figure 4: The evolution of the average amount of resources of non-collaborators in agent societies living in environments with different risk levels.

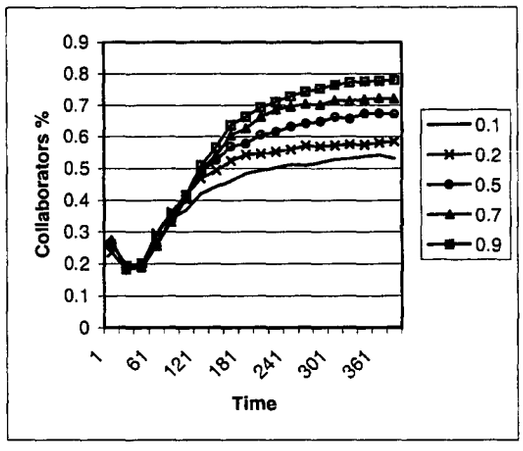


Figure 5: The evolution of the average percentage of collaborators within the agent societies living in environments with different risk levels.

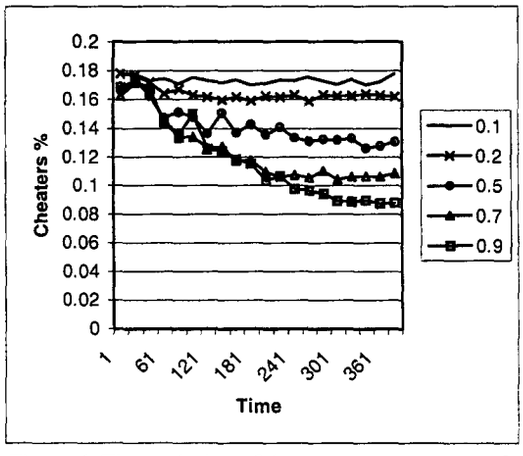


Figure 6: The evolution of the average percentage of cheaters within the agent societies living in environments with different risk levels.

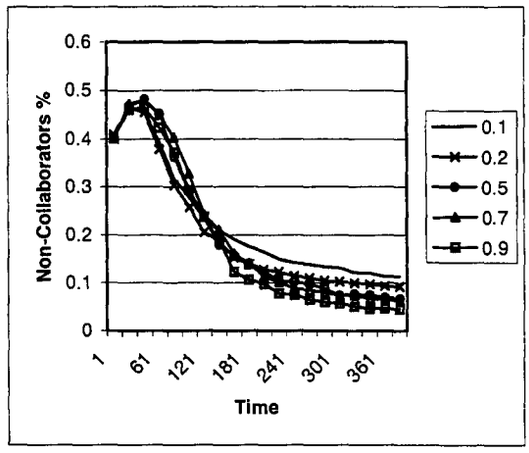


Figure 7: The evolution of the average percentage of non-collaborators within the agent societies living in environments with different risk levels.

The figures show that the level of collaboration increases in all conditions. After the first generation (i.e., around 60 time units) the level of collaborators increases steadily until it stabilizes (above 50%). In the case of cheaters and non-collaborators there is a corresponding decline to stabilization at below 18% for cheaters and below 12% for non-collaborators.

The figures show that the stable level of collaborators is lower in low risk conditions than in high risk conditions, and that levels of cheaters and non-collaborators are higher in low risk conditions than in high risk conditions. This indicates that the agent societies living in a high risk environment are more likely to achieve high level of collaboration than those which live in low risk environments. We note also than in the high risk environments it is more likely that the population dies out than in low risk environments.

3.3 The complexity of communications

We analysed the complexity of communications by measuring the average length of communication processes within the whole society.

Figures 8 shows the evolution over time of the communication complexity in the whole society.

The figure shows that there is no clear ordering of the stable levels of communication complexity as a function of the level of environmental risk. The same is true when each of the three agent strategies is examined independently.

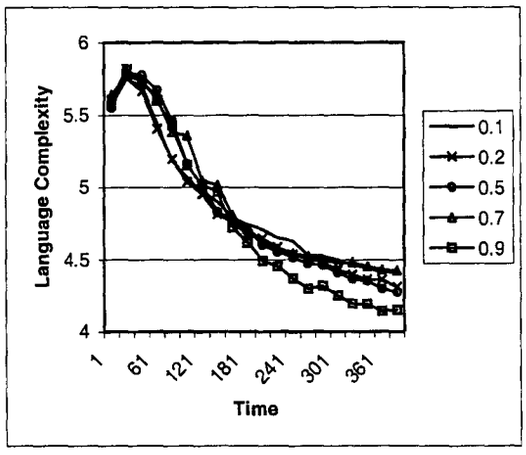


Figure 8: The evolution of the average communication complexity within the whole agent societies living in environments with different risk levels.

3.4 Collaboration and communication complexity

First we analysed the correlation between the levels of collaborators, cheaters and non-collaborators and the average complexity of communications within the society. These correlations are shown in Tables 1 – 3.

Risk	0.1	0.2	0.5	0.7	0.9
Correlation	-0.94	-0.98	-0.99	-0.99	-0.99

Table 1: The correlation between the average percentage of collaborators and the average complexity of communications within societies living in environments with various risk levels.

Risk	0.1	0.2	0.5	0.7	0.9
Correlation	0.15	0.74	0.93	0.95	0.97

Table 2: The correlation between the average percentage of cheaters and the average complexity of communications within societies living in environments with various risk levels.

Risk	0.1	0.2	0.5	0.7	0.9
Correlation	0.97	0.98	0.99	0.99	0.99

Table 3: The correlation between the average percentage of non-collaborators and the average complexity of communications within societies living in environments with various risk levels.

These results indicate that the proportion of collaborators is strongly negatively correlated with the complexity of communications at all risk levels. In the case of cheaters we see that their proportion is moderately positively correlated with the average complexity of communications at low risk levels, and that the correlation gets much stronger for high risk levels. In the case of non-collaborators their percentage is strongly positively correlated with the average communication complexity at all risk levels. These results together suggest that those who collaborate tend to communicate in shorter sequences, while those who cheat or do not collaborate are likely to use longer communication sequences.

To analyse this suggestion directly, we examined how communication complexity evolves in the three different groups at given risk levels. Figures 9 and 10 show two examples for risk levels 0.2 and 0.9.

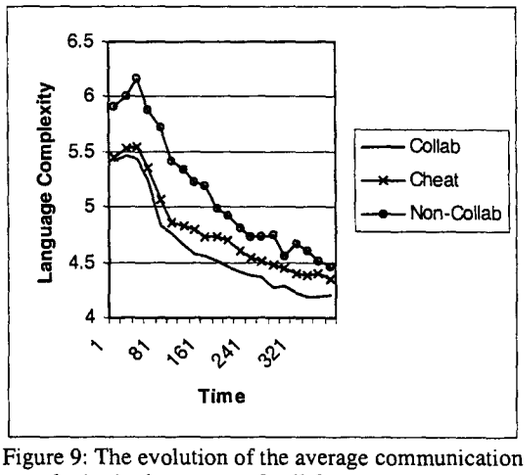


Figure 9: The evolution of the average communication complexity in the groups of collaborators, cheaters and non-collaborators living an environment characterized by risk level r = 0.2.

These figures confirm the suggestion that those who collaborate are likely to use less complex communication between themselves, and those who do not collaborate use more lengthy communication processes.

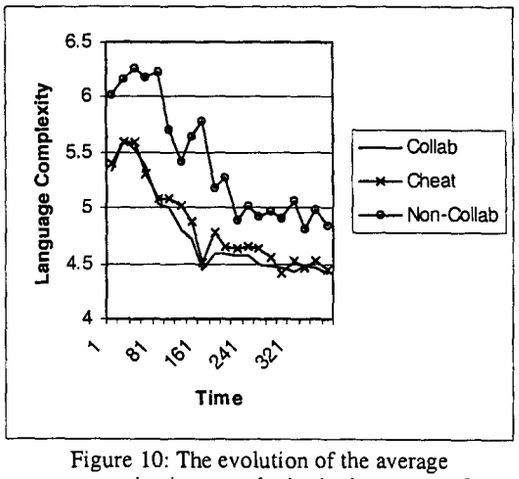


Figure 10: The evolution of the average communication complexity in the groups of collaborators, cheaters and non-collaborators living an environment characterized by risk level r = 0.9.

4 Discussion

First, we interpret our results from the more general point of view of evolution of collaboration and communication in societies of individuals. Second, we discuss the implications of the presented results for agent worlds and multi-agent systems.

4.1 Evolution of collaboration and communication in societies of individuals

Under the assumptions of our model society, cooperation can thrive and its frequency increases with environmental risk, while both cheating and non-cooperation decline. The increase in cooperation with environmental risk probably comes about because cooperation can be crucial for survival when resources are very low and/or provides particularly large rewards (compared to cheating) when resources are very high. This is because while cheating is profitable in the short term (for a few interactions), in the longer term cheaters fail to find other agents who will interact with them. The fact that population size and average resource level decline as risk increases supports the conclusion that cooperation becomes increasingly advantageous in difficult or harsh environments, as measured by risk.

The prediction that cooperation is more likely in risky environments can be tested in animal societies, in human experimental groups and in the real world of human social and economic behaviour, at the level of both individuals and of groups such as firms and nations. For example, this prediction might help to explain the phenomenon of increased feelings of community during wartime. It also suggests that cooperation might be enhanced by increasing the risk or complexity of the problem at hand. Although there may also be costs associated with increasing risk, if perceived risk increased while objective risk remained unchanged then cooperation might be enhanced without cost. However, as perceived risk increases the population of those willing to participate is likely to decline. A possible application area here is communication on the Internet, although there would be ethical issues involved in deceiving users about the risk or complexity of the Internet environment.

In contrast to its effect on cooperation, environmental risk in the model had no clear effect on the length of communication. This may have been because the model language was too simple, varying between only 4 and 6 elements at the outset. For a richer language we predict that communications will be shorter as risk increases since: (1) there is a positive correlation between collaboration level and risk, and negative correlations between cheating and non-collaboration levels and risk (Figures 5-7), and (2) there is a negative correlation between collaboration level and language complexity, and positive correlations between cheating and non-collaboration levels and language complexity (Tables 1-3).

Those who collaborated had shorter communication strings than those who cheated or failed to collaborate. This is because if a collaborator meets an agent for whom it has a memory biased towards collaboration then it has higher probabilities of production for positive communication symbols and therefore moves more quickly (i.e., with fewer communication steps) into an interaction that is likely to be collaborative. Thus collaborating agents, by positive feedback, build an increasingly cooperative relationship with each other, in a manner analogous to that described by Roberts & Sherratt (1998). The direct complement of this process is that meeting a past cheater for which an agent has a memory increases that agent's likelihood of cheating in the present interaction after a longer series of communication symbols (see sub-section 2.3 on the communication process).

Collaboration thus brings with it the bonus of a saving on communication effort. Such effort may be trivial, but it may also be considerable, as in some forms of human negotiation. The prediction that collaboration simplifies the communication process (compared to both cheating and avoiding interaction) can be tested in the scenarios already described for examining the relationship between cooperation and risk.

4.2 Collaboration and communication in agent worlds

We see two directions of implications of our work in the context of agent worlds and multi-agent systems.

First, our results indicate that appropriate setting of the environmental risk factors of an agent world can determine to a significant extent the level of collaboration within the agent world. This may have applications in the design of multi-agent systems where the developers wish to achieve some desired mix of collaborative/non-collaborative behavioural patterns that fits to the objective of the system. It is important to note that pure collaborative behavior in an open agent world may pose significant risks to the proper working of that world, as malignant agents may appear, and may abuse the default benevolent behavior of other agents. This means that some level of non-cooperative behavior should be allowed in an open agent system (Sherratt and Roberts 2001).

Second, our results suggest that it is possible to predict the expectable level of collaboration and communication complexity in an agent world, if enough information is available about the environmental risk factors characterizing this world. Such predictions can form the basis for checks of the validity of risk factor assumptions, and for corrective actions aimed to keep the agent world within the desired range of macro parameters.

References

R.D. Alexander. *The Biology of Moral Systems*. Aldine de Gruyter , New York, 1987.

R. Axelrod. *The Evolution of Cooperation*. Basic Books, New York, 1984.

R. Axelrod, & W.D. Hamilton. The evolution of cooperation. *Science*, 211:1390-1396, 1981.

S.D. Boon and J.G. Holmes. The dynamics of interpersonal trust: resolving uncertainty in the face of risk. In: R.A. Hinde and J. Groebel (eds.) *Cooperation and Prosocial Behaviour*, pp. 190-211. Cambridge University Press, Cambridge, 1991.

R. Dawkins. *The Selfish Gene*. Oxford University Press, Oxford, 1976.

M. Enquist. Communication during aggressive interactions with particular reference to variation in choice of behaviour. *Animal Behaviour*, 33:1152-1161, 1985.

O. Leimar. Reciprocity and communication of partner quality. *Proceedings of the Royal Society of London Series B: Biological Sciences*,264:1209-1215, 1997.

O. Leimar and P. Hammerstein. Evolution of cooperation through indirect reciprocity. *Proceedings of the Royal Society of London Series B: Biological Sciences*, 268:745-753, 2001.

J. Maynard Smith 1974 The theory of games and the evolution of animal conflicts. *Journal of Theoretical Biology*, 47:209-221.

M.A. Nowak, and K. Sigmund. Evolution of indirect reciprocity by image scoring. *Nature*, 393:573-577, 1998.

R.L. Riolo, M.D. Cohen, and R. Axelrod. Evolution of cooperation without reciprocity. *Nature*, 414:441-443, 2001.

G. Roberts. Competitive altruism: From reciprocity to the handicap principle. *Proceedings of the Royal Society Series B: Biological Sciences*, 265:427-431, 1998.

G. Roberts and T.N. Sherratt. Development of cooperative relationships through increasing investment. *Nature*, 394:175-179, 1998.

M. Schillo, P. Funk and M. Rovatsos.Using trust for detecting deceitful agents in artificial societies. *Applied Artificial Intelligence*, 14:825-848, 2000.

T.N. Sherratt and G. Roberts. The role of phenotypic defectors in stabilizing reciprocal altruism. *Behavioural Ecology*, 12:313-317, 2001.

R.L. Trivers. The evolution of reciprocal altruism. *Quarterly Review of Biology*, 46:35-57, 1971.

Stochastic Simulation of Inherited Kinship-Driven Altruism

Heather Turner; Dimitar Kazakov
Computer Science Department
University of York
{hrt103,kazakov}@cs.york.ac.uk

1 Introduction

The aim of this research is to assess the rôle of a hypothetical inherited feature (gene) promoting altruism between relatives as a factor for survival. The two main goals are, firstly, to replicate the phenomenon of altruism, which has been observed in nature, and show that the proposed mechanism leads to altruistic individuals being selected by evolution. Secondly, the research aims to provide an implementation of a Multi-Agent System (MAS) employing a model of natural selection, which is different from the one commonly used in Computer Science (Goldberg, 1989), and, hopefully, closer to the one existing in nature.

Altruism can be defined as selfless behaviour, action that will provide benefit to another at no gain to the actor himself, and possibly even to his detriment. In *kinship-driven altruism*, this behaviour is directed between individuals who are related. Hamilton (1964a) introduces an analytical model, in which altruistic behaviour towards relatives is favoured by evolution, provided that the amount of help that an individual bestows on relatives of a given distance is appropriately measured.

Both MASs (Wooldridge and Jennings, 1995) and Genetic Algorithms (GAs) (Goldberg, 1989) can be used effectively to simulate the interaction of a population that evolves over a period of time. A MAS allows study of the interactions at the level of the individual, while a GA is a better tool for generalisation over an entire population. In a GA, no distinction is made between individuals with the same *genotype* (i.e., inherited features), whereas in a MAS these are represented by different *phenotypes*, or set of observable characteristics resulting from the interaction of each genotype with the environment (Thompson, 1996). The use of MAS with large populations is limited by the requirement for extra resources to represent individual phenotypes. In a GA, the individual is anonymous, so there is no capacity to "zoom-in" on its behaviour, but in contrast, there is the possibility of scaling up to consider a much larger population, which may be statistically more relevant.

The GA uses a fitness function to estimate how well each individual will fare in the future and uses this to influence the likelihood that they survive to subsequent generations. A MAS uses information about the current position of an individual in the environment, and taking into account its internal state, considered to be the cumulative result of its actions and experiences in the past, determines its actions. In a GA, the population size is fixed, and during each system cycle, individuals may be selected to mate (and be replaced by their descendants) or they pass to the next generation. The anonymity of each individual is suited to the probabilistic selection afforded to this algorithm, and the resulting possibility that clones of an individual be produced in future generations. Without this anonymity, in a system that 'tracks' the behaviour of individuals through the generations, complications could arise on cloning. Attachment of energy values becomes difficult if the probabilistic freedom is to be maintained without producing a system that can produce and destroy energy at will. In a MAS, the population size is not explicitly constrained, and the internal state of an individual determines its lifespan. A system cycle will not generally represent an entire generation, as individuals may survive for many cycles. Table 1 summarises the main differences between the GA and MAS models of natural selection.

We combine features of each approach to produce a more scalable, personality-driven system without a modelled spatial dimension. The probabilistic nature of all events and the high level of abstraction typical for the GA are preserved. However, the description of each individual consists of a set of inherited features (genome) along with a—very abstract—description of the actual organism (phenotype). The internal state of each individual is changed by the interaction with a very simple, abstract, environment, in which both the selection of an individual's action and its outcome are modelled as probabilistic functions. This permits to avoid the use of an explicit fitness function, and describe the survival of an individual directly as a probabilistic function of its internal state (e.g., current energy levels).

Our system is designed to simulate a population in which some of the individuals are carriers of a gene forc-

Table 1: MAS vs GA simulation of natural selection

Feature	MAS	GA
Representation of individuals	genotype + phenotype	genotype only
Survival of individuals	deterministic, based on the lifetime interaction with environment	probabilistic, based on genotype's fitness
Population size	unlimited	fixed
Environment resources	limited capacity	use bounded by max. pop. size
Preservation of energy	enforced	not considered

ing them to share their energy with the individuals they meet in proportion to the degree of kinship (i.e., number of shared genes). The exact sharing policy is subjected to selection and studied. The altruistic gene is passed with a certain probability from parent to child. Food consumption results in increased individual energy level. Falling below a certain energy level means death. An encounter of two individuals of the same species could result in the creation of an offspring if their energy levels are sufficient. The initial energy level of the offspring is subtracted from that of the parents.

This research uses the hybrid platform described above to study from a different angle an extended version of some of the experiments with kinship-driven altruism performed by Barton (2001). As in Barton's study, natural selection works to produce an optimum sharing function when it is left to vary.

2 Altruism and Darwinian Theory

The possible evolution of a selfless gene is an interesting area of study as it does not necessarily seem intuitive that an individual should value the survival of another to the extent of causing detriment to itself (perhaps by decreasing his own chance of mating or survival) in order to help the other. It would be in contrast to the classic Darwinian theory of natural selection, according to which selfish individuals would always take the upper hand, and eliminate altruists, as the behaviour of the latter would by definition hinder their reproductive success. There is evidence however, as Hamilton (1964b) illustrates, that many species in nature exhibit this altruistic trait. Neo-Darwinian theory (Watson, 1995) attempts to provide an explanation with the idea of 'inclusive fitness', and the hypothesis that natural selection works not at the level of an individual, but on each individual gene. Many individuals can carry copies of the same gene, and if these individuals could identify one another, it would be possible for them to aid in the process of natural selection over that gene by attempting to secure reproductive success and the passing of this gene to the next generation. The evidence provided by Hamilton suggests that nature has evolved to recognise that it is likely for close relatives to have similar genetic makeup. In Hamilton's model, the degree of kinship is quantified, and it can then be used to determine how much help an individual can bestow on a relative, at detriment to itself and yet still be likely to benefit the inclusive fitness, the 'fitness' of the gene.

Barton (2001) used a MAS to model a population of individuals who behaved altruistically competing in an environment with a population of the same size that was not altruistic. His MAS used GA principles by associating genes with each individual in an attempt to find optimum solutions for variables used in his simulations. In some of his experiments, it was the sharing population that prevailed, in others, the non-sharing population over-ran the environment. He quotes 'Gause's Competitive Exclusion Principle', stating 'no two species can coexist if they occupy the same niche', and hypothesises that given the limitations of his simulated system, his competing populations are likely to 'end up having the same, or very similar, niches'.

In the MAS he uses, there are agents to represent food and the individuals of each population. The environment is represented on a grid with varying terrain that could restrict movement, or provide water as sustenance to fulfil 'thirst,' one of the 'drives' that describe the internal state of an agent in a given cycle. Each agent uses the values of its drives, its immediate surroundings and some deterministic rules to make life choices in each cycle.

3 Design

The system we have implemented to investigate altruistic behaviour combines features used in a MAS and those used in a GA. Rather than providing co-ordinates for the position of each individual in the system, we model encounters with food (energy) and other individuals probabilistically, reflecting the likelihood that these would occur in a given cycle. We do not constrain the population size, thus permitting easier comparisons with Barton's work (Barton, 2001). We stem the growth of our population by increasing the probability of random death as the individual ages. The individuals in our implementation retain a portion of genetic material, encoding their behaviour, and sharing policy, and thus allowing evolution of optimum policies. The diagram provides the proposed environmental interaction module for our system. Each individual stores as its phenotype the value of its sex drive, its hunger (or energy level), age and the probability of survival. These values are updated in each system cycle. Figure 1 contains an outline of the proposed simulation, where individual boxes have the following functions:

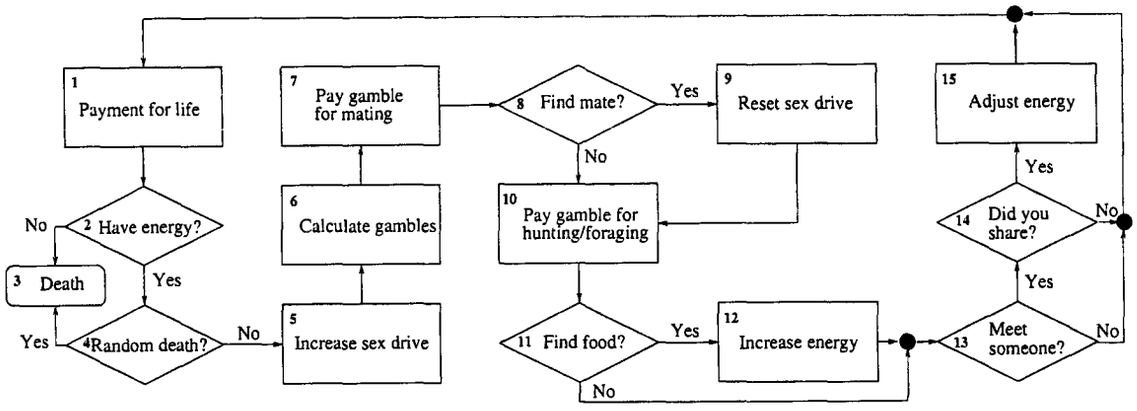


Figure 1: Simulation outline

1. Make a payment of energy to the environment (energy expended to survive generation).
2. If all energy is used up, one dies.
3. Individual has 'died' and is therefore removed from the population.
4. Random death occurs with some probability for each individual (probability increases with age).
5. Increase sex drive, an thus priority of reproduction.
6. Genetic material encodes a function to determine behaviour based on the values of the drives. This function produces "gambles" dictating how much of the available energy to expend in search for a mate or food, if any.
7. The gamble for mating is 'paid' to the system.
8. A probability distribution over the gambles is used to choose individuals for the mating pool(s). Pairs selected at random from the mating pool are deemed to have 'met' with some probability. Each must satisfy certain energy requirements. The probability that they mate is calculated from their sex drives and determines whether or not they actually 'mate'. On mating, new individuals are created from clones of the genetic material, and by resetting non-genetic parameters. Each parent contributes energy for sharing equally amongst the offspring. The clones undergo crossover producing two children to be included in the population for the next cycle.
9. The sex drive of the individuals who mated successfully is reset.
10. The gamble for hunting/foraging is 'paid' to the system.
11. A probability distribution based on the gamble determines how much energy an individual receives. For a gamble of zero, the probability that an individual receive any energy should be very low.
12. Energy level is increased by the amount of food found.
13. Pairs are further selected from the population, and with some probability are deemed to meet.
14. If the better fed of the pair is an altruist, they decide to share as per his genetically encoded sharing policy.
15. The energy of each individual is then adjusted as appropriate.

4 Experiments and Evaluation

The gambling policies of the tool described in the previous section were implemented as stochastic functions of the current needs of the individual. A sigmoid function was used to provide a nonlinear mapping from gamble to average win μ. The actual win of each gamble was generated according to a Gaussian distribution $G(\mu, \sigma)$ where the ratio σ/μ was kept constant for all μ, to ensure that only a very small, fixed proportion of the wins were negative; these, when generated, were reset to zero.

In the simulation, spatial phenomena (food discovery, encounter with another individual) are represented as random processes with a certain probability. In this case, physical distance between individuals is ignored, and the encounter of each pair is equally probable.

The so specified tool was implemented in C++, and used to study the influence of several factors on the evolution of altruistic behaviour. In all cases, the evaluation assesses whether the hypothetical altruistic gene is selected by evolution, and study the circumstances in which this happens.

Degree of kinship Individuals may (1) have a complete knowledge of their genealogy (*Royalty* model), (2) estimate the degree of kinship according to the presence of some inherited visible indicators (*Prediction*), or (3) not have this information available (*Unknown*).

Type of sharing function Three social models are considered. *Communism* equalises the energy levels of two individuals with the same genome. *Progressive taxation with a non-taxable allowance* is a simple linear function with a threshold: $y = \alpha(x - \theta)$ for $x > \theta$; $y = 0$ otherwise. *Poll tax* defines an altruistic act between two individuals as an exchange of a fixed amount of energy pt set in the genes of the donor, which does not depend on the energy level of either individual. The above descriptions corresponds to the case of sharing between two individuals with the same set of genes. In all other cases, the actual amount given is reduced in proportion to the difference between the two individuals' genomes, as derived from the perceived degree of kinship.

All combinations of the above two factors have been studied by running three times each of the nine possible experiments (see Figure 2). For technical reasons, the Royalty model of kinship stored only relatives up to first cousins (inclusive). In the predictive mode, all genes but (the altruistic) one were made visible to other individuals, and the difference between two sets of genes was defined according to a simple linear metric.

All parameters of the sharing functions (α, θ), resp. pt were initially set at random, and let to evolve. When employing the Unknown model of kinship, the most optimistic assumption was made, i.e., the donor treated the aid receiver as an identical twin brother.

In the experiments, all individuals carry a gene defining them as either selfish or altruistic. Simply counting the individuals carrying either gene is a good measure of the altruism in the population only in a communist society. In the other two cases, individuals, which are nominally altruistic, can have their sharing parameters set in a way, which reduces the effects of altruism to an arbitrary low level, e.g., α or $pt \to 0$, $\theta \to \infty$. In these cases, the ratio of what is given to what is actually owned by the individual, integrated over the whole energy (food) range, is considered a more appropriate measure. The idea in the case of progressive taxation is shown in Figure 2 where a nominally altruistic individual is assigned a degree of altruism given by the ratio of the filled triangle and the square made of the ranges of energy owned and exchanged.

The graphs in Figure 2 are self-explanatory. In brief, the use of either perfect knowledge of the degree of kinship or a sharing function based on progressive taxation ensures that a substantial level of altruism is selected and maintained in the population. In the remaining cases, altruists perish faster when less kinship knowledge is available. The population size remains the same in all cases, and is given by the amount of food supplied. A representative example of the way in which the population size evolved is shown in Figure 4 on the case of Royalty with Progressive Taxation.

Initial ratio between altruistic and selfish individuals To study the influence that the initial proportion of altruistic to selfish individuals has on the levels of altruism selected by evolution, the Royalty with Progressive Taxation experiment was run with several initial values for this ratio. The results in Figure 3 show that the system reaches a dynamic equilibrium which, in the cases shown, does not depend on the initial point.

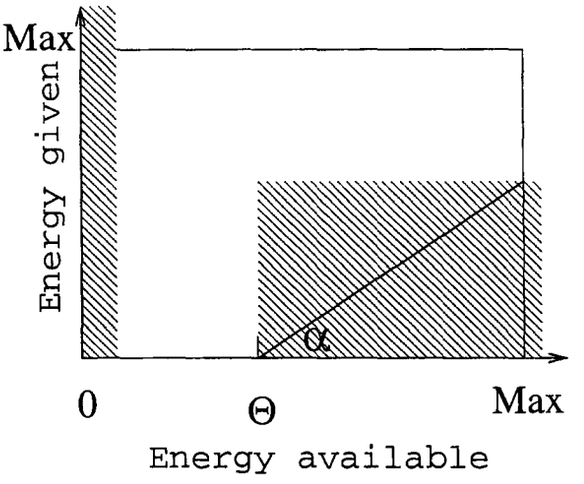


Figure 2: Measure of altruism

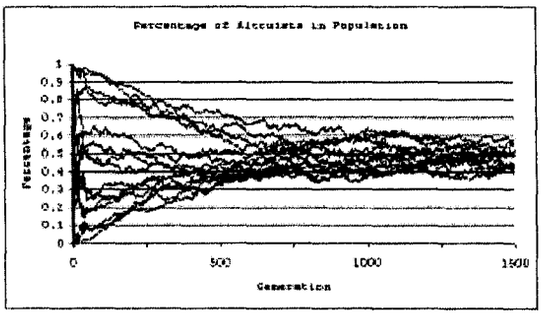
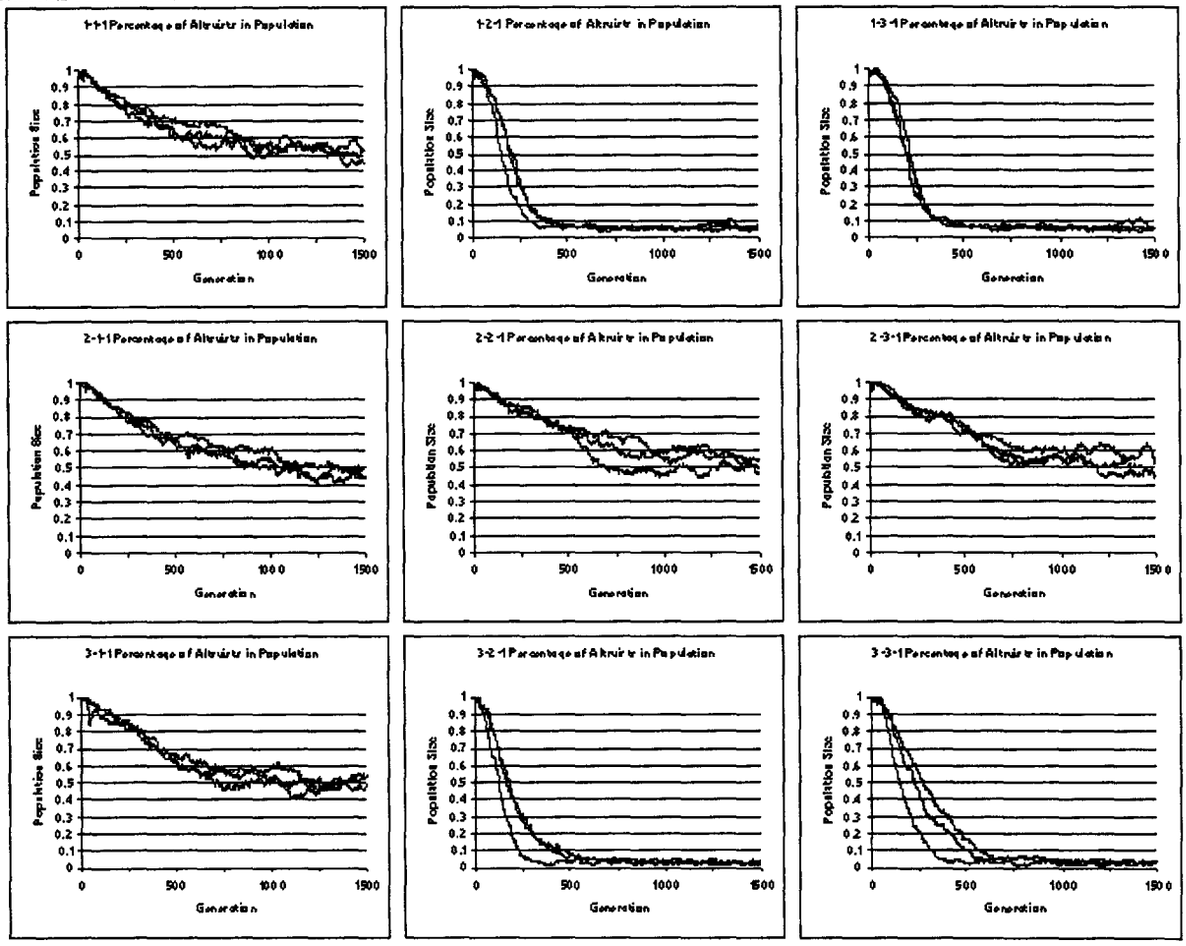


Figure 3: Percentage of altruists in the population with respect to initial levels (1=100%)

5 Discussion

Both goals of this research, as stated in Section 1, are successfully met. The proposed algorithm has been implemented, and altruism has, indeed, been shown to be selected and maintained by evolution in a number of cases. No direct comparison with the Barton's work could be made as his detailed results were not available in a suitable form. However, a few main points can be made. Firstly, it has been confirmed that the policy of progressive taxation produces more altruists than communism. An additional policy (poll tax) was studied in this research, which also introduced the new dimension of 'knowledge of the degree of kinship' in the experimental setup. Unlike Barton's, these experiments produced populations of

Table 2: Percentage of altruistic individuals in the population (1=100%). (Columns, from left to right: Royalty, Prediction and Unknown models of kinship recognition. Rows, top to bottom: Communism, Progressive Taxation and Poll Tax sharing functions.)



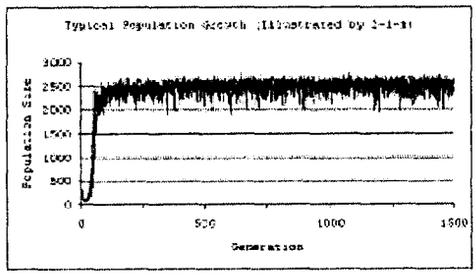


Figure 4: Evolution of population size

virtually the same size. Barton treats altruists and non-altruists as two different species, which in turn results in one species taking completely over the other one. In our results, there are several cases in which a balance between altruists and selfish individuals is maintained.

Altruism is a demonstration of the mechanisms on which natural selection is based. Note that this work does not aim to imply the existence of such gene in reality, and indeed nothing of the said above would change if one assumed altruistic behaviour being inherited not as a gene, but through upbringing.

There is interest in the use of natural selection in artificial societies. This research should bring the implementation of natural selection in artificial societies a step closer to the original mechanism that is copied. The authors' expectations are that the natural selection incorporating altruism would be suitable in cases, when the task is to produce an optimal population of agents rather than a single best individual, in situations when the knowledge about the performance of the population is incomplete and local.

The software described here may also represent a useful tool for the simulation of natural societies and give an interesting insight in their inner making, although this would be to experts in the relevant fields to judge.

The main characteristics of the model of altruism discussed here, namely, 'inherited' and 'kinship-driven', also mark the limits of its reach.

Firstly, the model does not allow changes in the altruistic behaviour of an individual within its lifespan. In fact, natural selection and individual learning are not perceived here as mutually exclusive. It is expected that, in many cases, combination of the two could be a successful strategy, where natural selection provides the starting point for the individual behaviour, which is modified according to the agent's personal experience. The actual technique employed at this second stage could be, for instance, based on game theory, where natural selection provides a suitable initial strategy. If individual behaviour is to be modified by a machine learning technique, natural selection could also provide it with a suitable bias. Research in this direction should be helped by the York MAS, currently under development, which supports natural selection among agents, as well as logic-based programming of behaviour and individual learning (Kazakov and Kudenko, 2001).

The second limitation of the model of altruism discussed here is that it does not discuss the case when agents can at will opt in and out of a society promoting altruism among its members. Since the names of many such societies draw analogies with kinship, e.g. 'fraternities' or 'sororities', in order to evoke the corresponding spirit of altruism (or 'brotherhood') in its members, the authors believe that also in this case the findings described in the paper would not be without relevance.

In comparison with logic-based approaches, this research makes one simple initial assumption, and attempts to see if altruism can be worked out from first principles. The actual behaviour of agents can be deterministic (and described in logic) or stochastic, that should not be of principle importance. On the other hand, no further background knowledge is assumed here—the agent's rules of behaviour are let to evolve, and not set in advance. In the future, comparisons with Hamilton's analytical model, and the evolutionary game theory point of view would also be worth exploring.

6 Future Work

It would also be interesting to extend the platform developed to implement different mating policies, so that pairs of individuals could be selected from a single mating pool or from separate mating pools into which individuals have previously been grouped according to their internal state: qrich meet (mostly) rich, poor meet poor, individuals with high sexual drive are grouped together, etc.

In addition, two environmental parameters, resource availability and the probability of meeting another individual, should be taken into account, and used to test the effectiveness of altruistic vs. selfish policy in various, and changing, environments.

An important and, potentially, non-trivial issue is the analysis of the content of the individuals' sets of genes and their evolution in time. In the case when the propagation of all genes is subject to simultaneous selection, one would have to study data sets, which are multidimensional–one dimension per locus plus an extra dimension representing time–hence difficult to visualise. One could expect that there would be a correlation between the genes selected in each locus, and that certain combinations might show a trend of dominating the population, which would form clusters around those points. Methods and tools for multivariate data visualisation with a minimal loss of information, such as those described by Schröder and Noy (2001), will be considered for the above task.

Acknowledgements

The second author wishes to expess his gratitude to his wife María Elena and daughter Maia for being such a wonderful source of inspiration.

References

John Barton. Kinship-driven altruism in multi-agent systems. Project report for a degree in Computer Science, University of York. Project supervisor: Dimitar Kazakov, 2001.

David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

W. D. Hamilton. The genetical evolution of social behaviour I. *Journal of Theoretical Biology*, 7:1–16, 1964a.

W. D. Hamilton. The genetical evolution of social behaviour II. *Journal of Theoretical Biology*, 7:17–52, 1964b.

D. Kazakov and D. Kudenko. *Milti-Agent Systems and Applications*, chapter Machine learning and inductive logic programming for multi-agent systems, pages 246–270. LNAI 2086. Springer, 2001.

Michael Schröder and Penny Noy. Multi-agent visualisation based on multivariate data. In *Working Notes of the Fourth UK Workshop on Multi-Agent Systems UKMAS 2001*. 2001.

Della Thompson, editor. *The Oxford Compact English Dictionary*. Oxford University Press, 1996.

Tim Watson. Kin selection and cooperating agents. Technical report, Dept. of Computer Science, De Montfort University, Leicester, 1995.

M. Wooldridge and Nick Jennings. Intelligent agents: theory and practice. *Knowledge Engineering Review*, 2(10), 1995.

EVOLVING PREFERENCES AMONG EMERGENT GROUPS OF AGENTS

Paul Marrow, Cefn Hoile, Fang Wang, Erwin Bonsma
BTexact Technologies; Intelligent Systems Laboratory,
Adastral Park, Ipswich IP5 3RE, UK
{paul.marrow,cefn.hoile,fang.wang,erwin.bonsma}@bt.com

Abstract

Software agents can prove useful in representing the interests of human users of agent systems. When users have diverse interests, the question arises as to how agents representing their interests can be grouped so as to facilitate interaction between users with compatible interests. This paper describes experiments in the DIET (Decentralised Information Ecosystem Technologies) agent platform, using evolutionary computation to evolve preferences of agents in choosing environments so as to interact with other agents representing users with similar interests. These experiments suggest a useful way for agents to acquire preferences for formation of groups for information interaction between users, and may also indicate means for supporting load balancing in distributed systems.

1 Introduction

Software agents have proved useful for representing the interests of individual human users (e.g., Klusch 2000). With multi-agent systems there arises the possibility of managing processes on behalf of large populations of human users. Associated with this is the problem of ensuring that users with common interests get to interact appropriately. This is the issue of group formation, part of the more general problems of cooperation and collaboration in multi-agent systems.

Group formation and the associated issues of cooperation and collaboration have proved relevant to much research in multi-agent systems (e.g., Jonker et al. 2000; Klusch & Sycara 2001; Wang 2002). Similar problems exist in robotics (e.g., Matarić 1997). One particular focus of research has considered how evolutionary algorithms can be used to adapt agent behaviour, and achieve collaborative or cooperative solutions (Haynes et al. 1995; Gordin et al. 1997; Moukas & Zacharia 1997). The use of evolutionary algorithms seems particularly appropriate in this context since they depend upon the interaction of many individuals for their success (Bäck et al. 2000).

In this paper we describe how an evolutionary algorithm can be used to adapt agent behaviour in the DIET (Decentralised Information Ecosystem Technologies) system (Marrow et al. 2001; Hoile et al. 2002), resulting in the emergence of groups of agents that share common interests. The DIET system implements large numbers of lightweight agents that can interact in a decentralised and extensible manner. The DIET system has been inspired by the interaction of organisms in natural ecosystems, and, inspired by the role of evolution in such interactions, the mechanism we use for group formation is the evolution of preferences for different environments. In the software agent context, an environment refers to the software environment an agent inhabits. In this context we assume that each environment exists on a single machine. For mobile agents, multiple environments may be on different machines. In a DIET network different environments maintain connections with each other in a "peer to peer" fashion.

It is well known that a degree of centralisation in peer to peer networks can improve the efficiency of functions such as indexing and retrieval (Oram 2001). However, existing strategies for centralisation often depend upon the existence of reliable, well known, central servers. Here we demonstrate the emergence of centralisation within a network of peers with no central servers. The dynamic approach described offers a compromise between the robustness and self-sufficiency of fully decentralised networks of transient peers with the efficiency of a centralised system.

The dynamic formation of communities of agents could be very important for the proper exploitation of computational and informational resources in future networks. The most rapid and effective interactions between agents typically are those that take place locally, between agents occupying a single environment. Accordingly we use an evolutionary algorithm to evolve preferences that lead to agents with common interests sharing the same environment.

Use of an evolutionary algorithm allows local interactions between agents to be taken advantage of in shaping the strategies used for despatch of agents to different environments over a sequence of iterative steps

(evolutionary generations). Working with two populations of agents, User agents, representing user interests, and Scout agents, searching out preferred environments, we use the evolutionary algorithm to evolve the preferences of Scout agents for environments in a network of multiple environments in which agents can exist. We show that the evolutionary algorithm can increase the effectiveness of Scout agents in locating environments that are suitable for information transfer with other agents representing common interests. This can provide a basis for the automatic formation of groups of users sharing interests.

We also consider how the results from the process of group formation indicate the robustness and flexibility of the DIET system. As well as explicit selection of agents through an evolutionary algorithm, we consider how characteristics of the DIET agent environment can stimulate a process of implicit evolution, that is evolution with respect to computational resource constraints, where computational efficiency is associated with survival. This could also be used to evolve agents that adopt computationally efficient behaviour.

2 The DIET Platform

The experiments presented here use the DIET (Decentralised Information Ecosystem Technologies) system (Marrow et al. 2001; Hoile et al. 2002), a software platform that has been developed to enable the implementation of multi-agent systems consisting of very large numbers of lightweight agents, under decentralised control, interacting in a manner inspired by natural ecosystems. The development effort within the DIET project (DIET 2001) is focused on providing an extensible framework for the exploration of ecologically inspired software solutions in an open agent platform.

2.1 Aims and Inspiration

Inspiration for the DIET system has come from natural ecosystems, where many living organisms and abiotic elements interact to produce a variety of emergent phenomena (Waring 1989). These biological systems have inspired the Universal Information Ecosystems initiative of the European Union (FET 1999), which addresses the issue of managing and understanding the complexity of the emerging global information infrastructure by looking at local and bottom-up interactions between elements of this infrastructure, in the manner of interactions between living organisms. Such local and bottom-up approaches may be expected to provide more flexibility, adaptability and scalability in response to changing circumstances than more top-down or centralised approaches. The DIET project forms part of the Universal Information Ecosystems initiative and hence the system design attempts to take these ideas into account.

2.2 Architecture

The DIET system is designed around a three-layer architecture (Marrow et al. 2001):

- *Core layer:* The functionality supported by the lowest layer is deliberately minimal, but designed to provide a robust and reliable service (Marrow et al. 2001; Hoile et al. 2002). It is here that the services and functions common to all agents are provided.
- *ARC layer:* Additional utilities are distributed along with the core layer, known as "Application Reusable Components" (ARC). These provide primitives that exploit the minimal functions of the core to support higher level activities common to many, but not all, applications. These include remote communication, multicasting and directory services.
- *Application layer:* This layer comprises additional data structures and agent behaviours for application-specific objectives.

The three-layer structure provides flexibility for implementing a variety of applications using the same core features of the agents and the software environment that they inhabit. It has been implemented in Java.

2.3 Core Layer

The core layer provides for Environments that are the basic units which DIET agents can inhabit. One or more Environment may be located with a DIET World, there being a single Java Virtual Machine for each World. The possibility exists for multiple Worlds in conjunction with multiple Java Virtual Machines, allowing for indefinite scaling up of the DIET system.

Each Environment provides minimal functionality to all agents, allowing for agent creation, agent destruction, local communication between agents, and initiation of migration between Environments. These basic functions have been designed so as to minimise the computational overhead required for their execution. The CPU time required for each function is not dependent upon the number of agents occupying the Environment, allowing efficient and rapid operation even in Environments inhabited by large numbers of agents. Operation of the DIET system is based upon these basic functions and the resulting local interactions between agents.

Local communication is central to local interaction between DIET agents. Local communication in this context involves the passing of messages and objects between two agents in the same Environment. The agent that initiates the communication must identify the agent that it wishes to contact – this can be done using a binary "name tag" associated with the target agent that is

randomly generated in its original Environment. In addition an agent has a "family tag" that indicates the group of agents to which it belongs. These are in consequence not necessarily unique, but may also be used for identification. Identification of agents by either of these methods is decentralised, being associated only with particular Environments, and thus scales well with larger systems.

Once a target agent has been identified, a Connection is formed between the two agents, allowing messages and/or objects to be passed between the two agents. Each agent has a message buffer that provides a space into which messages can be received. More information about local communication is given by Hoile et al. (2002). Remote communication, that is, communication between Environments, is also possible. The core layer provides only agent migration at the Environment level. Key functions associated with remote communication are provided in the ARC layer.

2.4 ARC Layer

The ARC layer provides for various extensions that can support remote communication between Environments, as well as other functions. These include "Carrier Pigeon" agents that can migrate to another Environment and then deliver a message by local communication to the intended target agent. Alternatively Mirror agents can be created in an Environment to support a communication channel to an agent in another Environment, via Carrier Pigeons that only the Mirror agent, and not the agent initiating the remote communication, interacts with. Remote communication via a Mirror agent looks like local communication to other agents in the Environment. Such means of remote communication allow for increased flexibility in interaction between agents distributed across multiple environments (Hoile et al. 2002).

2.5 Applications

Based on the functionality provided by the core layer and the ARC layer, applications can be developed based on the DIET platform, with application-specific code situated in the third, application, layer. Applications can also take advantage of visualisation and interactive control software that is being developed (van Lengen & Bähr 2001). The basing of application development on this architecture centred on local interactions between agents makes the DIET system particularly appropriate for the study of phenomena emerging from local interactions. We now go on to do this in the context of emerging preferences for environments supporting co-location for information sharing.

3 Experiments

We seek to generate emergent phenomena among agents in getting them to evolve preferences for particular environments (that are DIET Environments). As such agents can represent the interests of human users, this may be a useful mechanism for automatically ensuring that users' interests in terms of environmental preferences are best served.

We consider a situation where human users of an information management system connect transiently to a peer-to-peer network in order to identify information resources that satisfy their requirements for information. We assume that each user has a "category of interest" that represents some topic that they are particularly interested in. Users that are interested in the same category of interest are assumed to be interested in the same set of information, but to only have access to a subset of that initially. We also assume that users are interested in finding other users with the same category of interest and sharing information with them. Each user creates a DIET Environment from which agents can be created to facilitate the user's requirements.

3.1 World, Environments and Links

The experiments take place in the context of a DIET World composed of multiple Environments as described above (section 2.3). Each Environment is distinct from others in terms of its distinctive signature provided by a hashcode. The Environment's hashcode is generated based on the Environment's address in the DIET World. A 32 bit hashcode is used, because a hashcode of this form can easily be acquired from all Java objects. But this form of hashcode can be replaced by one of many other hashing schemes if required (see e.g. NIST 2001).

In our experiments Environments are connected in a peer network. This network is formed by choosing pairs of Environments at random, and then creating neighbourhood links between them. Such links are uni-directional, but link formation can be repeated to give the effect of a bi-directional link. This process is repeated until each Environment has on average four links. This level of connectivity is intended to approximate the connectivity of a fully decentralised peer network. The existence of such links between Environments allows agents to migrate between those Environments. Figure 1 illustrates what such an agent network might look like.

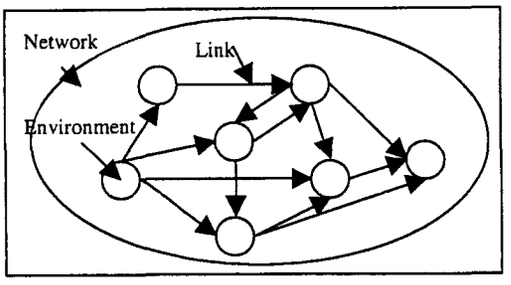


Figure 1: A possible DIET peer network

3.2 Agents

The experiments depend upon two populations of agents: User agents and Scout agents. These agents are lightweight and use the minimal functions supplied by the DIET platform. User agents represent human users and deploy and evolve Scout agents, as well as collecting information from Scout agents. Scout agents explore multiple Environments and carry out the activities needed to form groups.

Only one User agent is associated with a particular Environment. The User agent remains in that Environment throughout each experiment. Each experiment starts with a number of User agents distributed, one at each of the Environments.

Each User agent creates a population of Scout agents, and evolves them independently of other populations of Scout agents. Having created these Scout agents, the User agent dispatches them into the peer network of Environments, where they may interact with other Scout agents and other User agents, before returning to interact with the User agent that created them.

3.3 Evolutionary Algorithm

Scout agents are bred by User agents. User agents seek to maximise Scout agents' success in locating other Scout agents with common interests. A Scout agent's preference for Environments is determined by a bitstring genome provided at birth. A Steady State genetic algorithm is used (Sarma & De Jong 2000), implemented using the Eos evolutionary and ecosystem platform (Bonsma et al. 2000). Tournament selection, two-point crossover, uniform mutation and random replacement operators are used in the algorithm. Random replacement allows Scout agents to adapt their expectation of success under changing conditions of informational and environmental availability.

When dispatching new Scout agents, the User agent uses tournament selection to choose parent genomes from its population, favouring genomes according to the success of the respective Scout agent in locating other satisfactory Scout agents.

The behaviour of each Scout agent depends upon a satisfaction or preference function that indicates the degree of satisfaction that the Scout agent has with an Environment. This satisfaction function employs two bitstrings of length 32, drawn from a binary genome containing 64 bits. These two bitstrings are known as the XOR_mask and the AND_mask. To determine the degree of satisfaction for a given Environment, the Environment's hashcode is XORed with the XOR_mask, and then ANDed with the AND_mask. The number of set bits (i.e. bits with the value "true") then indicates the degree of satisfaction with the Environment. This preference function can then be evolved, in order to generate different orderings of preferences for Environments.

Scout agents are initialised with a success of zero. New generations of Scout agents are generated by the recombination of the genomes of two "parent" Scout agents, resulting in two new Scout agent genomes. Two new Scout agents result, that are released into the User agent's local Environment. From this point they carry out a three-phase life cycle (described below). If they complete this life cycle, and return successfully to the originating Environment, an existing member of the population of Scout agents based in that Environment is replaced at random by the genome of the returning Scout agent. In this way Scout agent preferences evolve over many generations in response to the conditions they encounter in different Environments.

3.4 Scout Agent Life Cycle

Having been created by User agents in their home Environment, Scout agents go through a life cycle that is divided into three phases: the Exploratory phase, the Sharing phase and the Reporting phase.

In the *Exploratory phase*, a Scout agent visits eight Environments in a random walk starting from the Environment in which it originated. At each Environment it requests four addresses of neighbouring Environments, selecting one of these at random for the next hop. These numbers are fixed across all experiments in order to allow comparison across peer networks of different sizes. After collecting the thirty-two Environment addresses in this way, the Scout agent applies its evolved preference function in order to calculate a satisfaction value for each of the thirty-two potential host Environments encountered. It then selects as a host the Environment with the address that gives it the highest satisfaction. Where several Environment addresses give the same satisfaction, the most recently visited is preferred. The Scout agent then enters the Sharing phase.

During the *Sharing phase* the Scout agent migrates to its preferred host Environment, and spends a pre-determined period interacting with other Scout agents in that

Environment – notifying them of its User agent's ID and category of interest, as well as noting the IDs and categories of interest represented by other Scout agents in that Environment. Then it moves to the Reporting phase.

In the *Reporting phase* the Scout agent migrates back to its originating Environment, notifies the originating User agent of its genome, and the number of successful encounters achieved. Scout agent success is measured according to the number of Scout agents encountered that were derived from different User agents (hence different Environments), but represented the same information category. So, a successful encounter in this context means an encounter with a Scout agent originating from other User agents that represent the same information category.

The Scout agent then destroys itself, but its genome may live on in that it may be selected to contribute to the next generation of Scout agents, according to its success in locating Environments that contain other Scout agents representing User agents with common interests. The use of tournament selection means that some Scout agents with success lower than the current Scout agent population average may contribute to the next generation, but they are less likely to, and Scout agents with higher success are more likely to be represented in the next generation. Tournament selection also ensures responsiveness to changing conditions.

3.5 Consequences of Agent Behaviour

The repetition of this three-phase life cycle over multiple generations will lead to changes in the numbers of Scout agents found in each Environment at each iteration (corresponding to a generation of the evolutionary algorithm.) The long-term solution should be a network of Scout agents clustered to different densities in different Environments, with average Scout agent preference for Environments evolved to a level that most effectively supports choice of Environments in which agents representing the same category of interest can interact. Accordingly, Scout agent success in achieving such interactions should be maximised. Such a network of information sharing agents may support several distinct groups of agents, as represented by the shaded and unshaded agents shown in Figure 2.

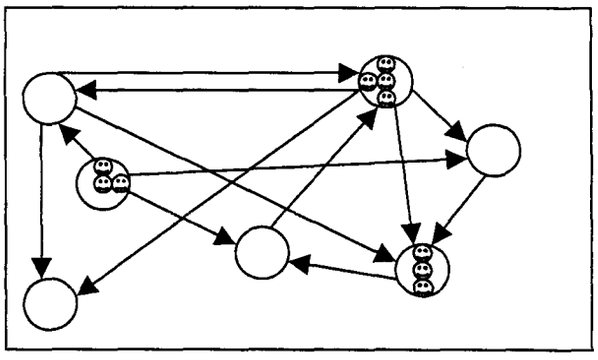


Figure 2: An example information network.

3.6 Experimental Conditions

The algorithm described above provided the basis for a series of experiments. In each experimental run we were interested in the effectiveness of the evolutionary learning among agents in stimulating co-location of Scout agents in appropriate Environments. For the sake of logging results, all Environments were hosted in parallel on a single machine. (However, there is no reason why they should not be hosted on multiple machines in parallel in the future.) To compensate for this lack of true parallelism, User agent search intervals, Scout agent waiting time, and overall run length were made proportionate to the number of User agents. A minute of CPU time was provided for the activity of each User agent. Each User agent began the simulation with a fixed category of interest, and a population of 100 Scout agents with random genomes (defining random preference functions). Initial experiments used the same category of interest for all User agents, but more than one category of interest can be used if required.

4 Results

Figure 3 shows the progress of a single experiment, involving thirty-two User agents. The number of Scout agents in each Environment changes over time due to the migration of Scout agents between Environments, as well as being due to the evolution of Scout agent genomes. The evolutionary algorithms are executed in real time by the parallel operations of all the User agents. For this reason results are shown against CPU time.

It is clear that one Environment in particular becomes the most popular, attracting the vast majority of Scout agents in the system. This distribution of Scout agents, with few agents in many Environments, and many in few, is the result of selection of Scout preferences for Environments based on interactions between Scout agents during the Sharing phase of their life cycle. This grouping of Scout agents could then be used to support more effective information exchange among the User agents in the system than was possible at the start of the experiment. It

indicates how this evolutionary approach could be useful in facilitating information interactions between the human users who have established such User agents.

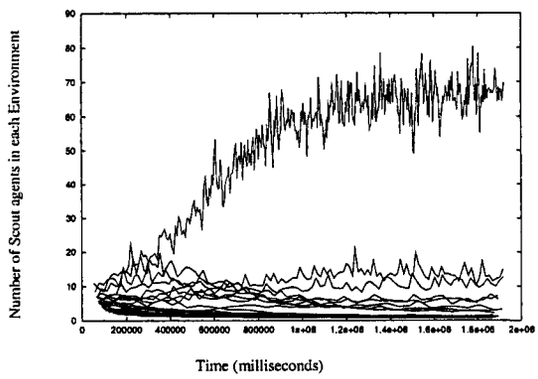


Figure 3: Environment population over time – 32 User agents.

Figure 4 shows how the phenomenon shown in Figure 3 occurs. Over time average Scout agent success increases, because the independent evolutionary algorithms converge to common Environmental preferences. This increases the Scout agent population density in certain Environments and hence increases Scout success.

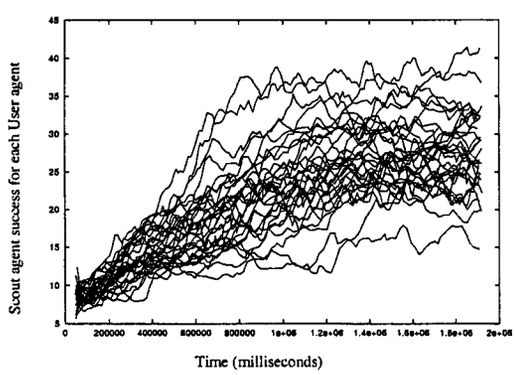


Figure 4: Average Scout agent success over time – 32 User agents.

If more User agents (and hence more Environments and more Scout agents) are involved, the system takes longer to evaluate where the appropriate Scout agents are, but still identifies them in the end. In Figure 5 we show the results of multiple runs of the algorithm designed to calculate the average (mean) value of Scout agent success. This is calculated after each 1 minute of CPU time has been used per User agent. We are interested to see whether use of the evolutionary algorithm has an effect on the average success of Scout agents.

Figure 5 shows that this occurs, in that average Scout agent success after one minute of CPU time is greater than the initial value (of zero). If the evolutionary algorithm is not used, so Scout agents have uniform preferences, average Scout agent success after one minute of CPU time, although non-zero, is constant irrespective of the number of User agents involved. If the evolutionary algorithm is used (represented by evolved preferences in the Figure), it is interesting that the average Scout agent success actually increases with the number of User agents, before declining at higher numbers of User agents. This suggests the benefit that the use of evolutionary techniques can offer among populations of agents in multi-agent systems, but also implies that very high numbers of User agents may make it more difficult for successful interactions between Scout agents to arise.

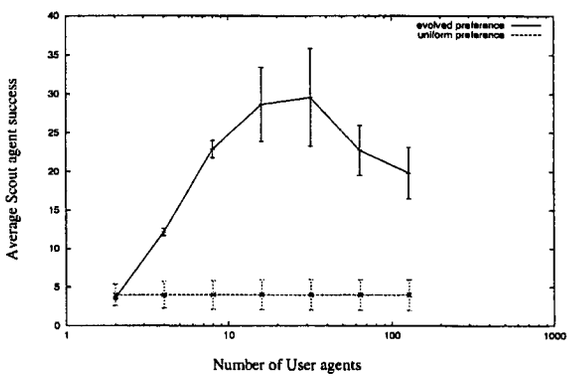


Figure 5: Average Scout agent success after one minute CPU time per User agent.

The results in Figure 3 shows that the evolution of Scout agent preferences for Environments can support convergence of many Scout agents to a single preferred Environment. When larger numbers of User agents are spread across more Environments, evolution of Scout agent preferences may result in several Environments supporting significant numbers of Scout agents in the long term (Figure 6). This does not indicate a failure to evolve to a sufficiently preferred solution, as comparison of the changes in average Scout success over time with this higher number of User agents with Figure 4 shows a similar change in success results although the final average is different (see Figure 7). In fact one Environment in Figure 6 ends up with significantly more Scout agents than all the others after the algorithm is run for some time. But this does not eliminate the several Environments that maintain persistent populations of Scout agents at somewhat lower levels. This is an inevitable source of the use of a random walk by Scout agents in locating Environments.

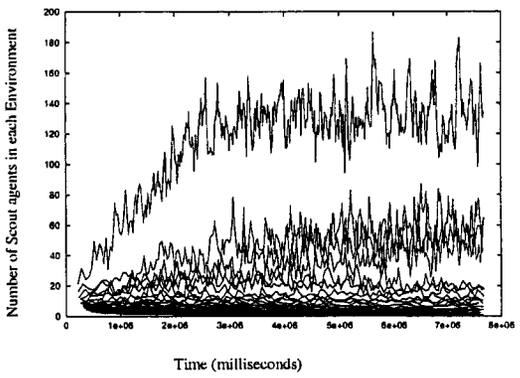


Figure 6: Environment population over time – 128 User agents.

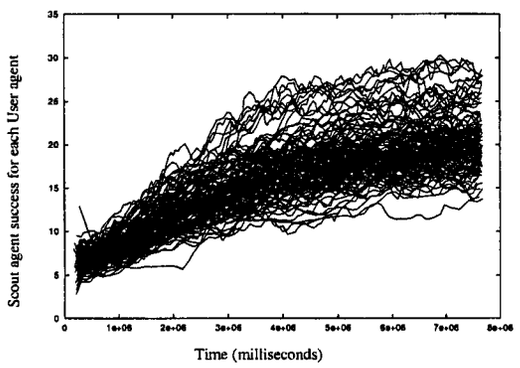


Figure 7: Average Scout agent success over time – 128 User agents.

5 Discussion

The experiments in evolving group formation that we have implemented using the DIET platform suggest that evolving agent preferences may be a useful means to tackle information management problems. Starting from a random initial assembly of users, agents quickly colocate according to the interests of their respective User agents. This facilitates more rapid and effective communications between User agents representing human users with common interests, and so shows the potential for application to more general peer-to-peer collaborative networks (Oram 2001). The experiments presented here are designed such that many User agents represent similar interests, but it would be possible to develop alternative scenarios where very many different interests were represented, and Scout agents spread out over many Environments.

While the results given above show convergence of the majority of Scout agents in the system to a single preferred Environment, it is likely that Scout agents will encounter a variety of Environments during exploration. The coexistence of agents in multiple Environments may provide additional robustness, since the loss of specific machines hosting some Environments is unlikely to eliminate all members of a specific agent community in a sufficiently large network. In addition agents persisting in such a diminished system will have the capability to evolve their preferences so as to adapt to the remaining Environments.

The experiments described above implement agents in multiple Environments in parallel on a single machine. It would be worthwhile to investigate larger networks of Environments and User agents with diverse categories of interest. Accordingly, further experimentation is planned, using multiple computers connected in a Beowulf cluster (Sterling et al. 1995). This should help reduce artefacts arising from thread sharing, and also permit the construction of larger peer networks.

The implementation of preference evolution on multiple machines may provide a means of using this algorithm for load balancing. This is because the use of multiple machines and hence system resources in parallel will provide the agents involved with the potential to evolve preferences and vary success at different rates in different machines. As a consequence, Scout agents will have the opportunity to switch between machines in order to improve their success rate in interacting with other Scout agents. While initial convergence of most Scout agents to a single Environment may result in a similar way to that in the experiments shown here, a consequence of this will be increased demands on one of the machines in the peer network. This will place constraints on the Environments hosted on that machine, restricting agent interactions. This may stimulate migration of Scout agents to other machines where system resources are less heavily in demand. The consequence of this will be a contrasting pressure on Scout agents to disperse over multiple machines, a kind of implicit evolution driven by available system resources.

This implicit evolution could be further used to develop groups of information sharing agents. The DIET platform provides the means to monitor the use of system resources by agents. Accordingly computational resource cost could be used to constrain the evolutionary algorithm so as to develop preferences appropriate to the machines (and/or resources) available at the time. In this way agents adopting computationally efficient behaviour can be evolved without explicit population management.

Acknowledgements

This work was carried out as part of the DIET (Decentralised Information Ecosystems Technologies) project (IST-1999-10088), within the Universal Information Ecosystems initiative of the Information Society Technology Programme of the European Union. We thank the other participants in the DIET project, from the Departmento de Teoria de Señal y

Communicaciones, Universidad Carlos III de Madrid, the Department of Electronic and Computer Engineering, Technical University of Crete, and the Intelligent and Simulation Systems Department, DFKI, for their comments and contributions. We also acknowledge the support of the Enterprise Venturing Programme of BTexact Technologies.

References

T. Bäck, D. Fogel, and Z. Michaelewicz (eds.). *Handbook of Evolutionary Computation.* Institute of Physics, Bristol, 2000.

E. Bonsma, M. Shackleton and R. Shipman. Eos – an evolutionary and ecosystem research platform. *BT Technology Journal* 18(4):24-31, 2000.

DIET project web site. http://pc-200.dfki.uni-kl.de:8080/DIET/public/index.html, 2001.

FET (European Commission IST Programme) Future and Emerging Technologies. Universal Information Ecosystems proactive initiative. http://www.cordis.lu/fetuie.htm, 1999.

M. Gordin, S. Sen & N. Puppala. Evolving cooperative groups: preliminary results. In: *AAAI-97 Workshop on Multi-Agent Learning*, 1997.

T. Haynes, S. Sen, D. Schoenefeld & R. Wainwright. Evolving a team. In: *AAAI Fall Symposium on Genetic Programming*, Cambridge MA, 1995.

C. Hoile, F. Wang, E. Bonsma, and P. Marrow. Core specification and experiments in DIET: a decentralised ecosystem-inspired mobile agent system. *1st International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS2002)*, Bologna (to appear), 2002.

C. Jonker, M. Klusch & J. Treur. Design of collaborative information agents. *Proceedings of 4th International Workshop on Cooperative Information Agents*, LNAI 1860. Springer, Berlin, 2000.

M. Klusch. Information agent technology for the Internet: a survey. *Journal of Data and Knowledge Engineering*, Special Issue on Intelligent Information Integration, D. Fensel (ed.), 2000.

M. Klusch & K. Sycara. Brokering and matchmaking for coordination of agent societies: a survey. In: *Coordination of Internet Agents: Models, Technologies and Applications*, A. Omicini, F. Zambonelli, M. Klusch & R. Tolksdorf (eds.). Springer, Berlin, 2001.

R.H. van Lengen & J.-T. Bähr. Visualisation and debugging of decentralised information ecosystems. *Proceedings of Dagstuhl Seminar on Software Visualisation*, Springer, Berlin, 2001.

P. Marrow, M. Koubarakis, R.H. van Lengen, F. Valverde-Albacete, et al. Agents in decentralised information ecosystems: the DIET approach. *Proceedings of the AISB'01 Symposium on Information Agents for Electronic Commerce*, pp. 109-117, York, 2001.

M.J. Matarić. Designing and understanding adaptive group behavior. *Adaptive Behavior* 4(1):51-80, 1995.

A. Moukas & G. Zacharia. Evolving a multi-agent information filtering solution in Amalthea. *Proceedings of Agents'97*, 1997.

NIST (National Institute of Standards and Technology) FIPS PUB 180-I. Secure hash standard. http://www.itl.nist.gov/fipspubs/fip180-1.htm, 2001.

A. Oram (ed.). *Peer-to-peer: harnessing the power of disruptive technologies.* O'Reilly Associates, Cambridge MA, 2001.

J. Sarma and K. De Jong. Generation gap methods. *Handbook of Evolutionary Computation*, C2.7, T. Bäck, D. Fogel, and Z. Michaelewicz (eds.). Institute of Physics, Bristol, 2000.

T. Sterling, D.J. Becker, D. Savarese, J.E. Durband, U.A. Ranawake & C.V. Packer. Beowulf: a parallel workstation for scientific computation. *Proceedings of 24th International Conference on Parallel Processing* 1:11-14, 1995.

F. Wang. Self-organising communities formed by middle agents. *1st International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS2002)*, Bologna (to appear), 2002.

R.H. Waring. Ecosystems: fluxes of matter and energy. In: *Ecological Concepts*, J.M. Cherrett (ed.). Blackwell Scientific, Oxford, 1989.

CONTROLLING THE ADAPTATION OF A POPULATION OF AGENTS

Philippe De Wilde
Intelligent and Interactive Systems Group
Department of Electrical and Electronic Engineering
Imperial College of Science Technology, and Medicine
London SW7 2BT, United Kingdom
p.dewilde@ic.ac.uk

Abstract

We control a population of interacting software agents. The agents have a strategy, and receive a payoff for executing that strategy. Unsuccessful agents become extinct. We investigate the repercussions of maintaining a diversity of agents. There is often no economic rationale for this. If maintaining diversity is to be successful, i.e. without lowering too much the payoff for the non-endangered strategies, it has to go on forever, because the non-endangered strategies still get a good payoff, so that they continue to thrive, and continue to endanger the endangered strategies. This is not sustainable if the number of endangered ones is of the same order as the number of non-endangered ones. We also discuss niches, islands. Finally, we combine learning as adaptation of individual agents with learning via selection in a population.

1 Populations of software agents

The growth of software has been found to be due to the increase in functionality of systems. This growth has become known as Software Evolution and has given rise to issues such as the Laws of Software Evolution.

Software that is subjected to evolution, E-Type, Kahen et al. (2001), resides in the real world and is subjected to unpredictable changes in the environment. The changes in the software environment are directly linked to the real world. Views, extracted from the real world are used as a model for the underlying structure of specifications and requirements, from which a program is constructed. Due to the real world being characterized with infinite assumptions and diverse features, there is no real benchmark that determines the 'correctness' of such a program. In order for the program come close to being correct, there must be conformity between the users' perception of the program functions and the real functions that the program offers.

Acceptability of a program or system is therefore dependent on the satisfaction of the user. So, a program or system can only be said to be 'correct' if the user is satisfied with it. As the environment continues to change, certain assumptions are no longer valid or acceptable. Hence, the program is forced to accommodate for this change, should it be required to maintain itself within the competition between software. The fact that technology advances also affects the users perceptions and expectations of the program, hence influencing the program's 'acceptability' level. When a program has finally gained acceptance, this is seen as a break through and this will provide the structure and foundation from which further improvements can be made and new technology developed. With competition of software in the market, all programs will strive to achieve higher payoff in order to satisfy users and remain in the market. Overall, all the software has to undergo continuous maintenance to the system so that the system payoff does not dip below the minimum level of payoff for survival.

Languages and information systems are vulnerable to a change in environment. However, they have the ability to mutate much faster than biological species. Although many languages and information systems may become extinct, the existing ones may be able to change so fast that they survive a change in environment. Germanic languages, for example, that for new words easily, are still very effective in describing computer systems. And a computer language such a C has shown a great capability for change when the environment changed to object oriented. It is essentially still around, under the name Java, in a new, network-based computing environment. If languages such as Forth, Algol, Ada, had been artificially kept alive, this would have slowed down the adaptation of the successful languages. This is the case for many information systems and types of software agents. Note that the evolution of single, large software projects has been studied in an evolutionary context similar to ours Kahen et al. (2001). Our approach, using populations, is more suitable for multi-agent systems than for programs operating in isolation.

Let us now study a population of software agents Jen-

nings (2000) that interact with each other. Each agent is uniquely determined by its code, just as a living organism is determined by its genetic code. Consider n different types of agents. At time t, there are $p_i(t)$ agents with code i in the population. Just as an agent is determined by i, a population is determined at time t by $p_i(t), i = 1, \ldots, n$.

The frequency of agent i in the population is

$$x_i(t) = \frac{p_i(t)}{\sum_{i=1}^{n} p_i(t)}. \quad (1)$$

Abbreviate $\sum_{i=1}^{n} p_i(t) = p$, where p is the total population. Denote the state of the population of agents by $\mathbf{x}(t) = (x_1(t), \ldots, x_n(t))$.

Now make the following basic assumptions Weibull (1995).

Assumption 1 (Game)
If agents of type i interact with a population in state $\mathbf{x}$, all agents of type i together receive a payoff $u(e^i, \mathbf{x})$.

Assumption 2 (Replicator Dynamics)
The rate of change of the number of agents of type i is proportional to the number of agents of type i and the total payoff they receive:

$$\dot{p}_i(t) = p_i(t)u(e^i, \mathbf{x}(t)). \quad (2)$$

The proportionality constant in (2) can be absorbed in u. These assumptions are discussed in the rest of this section.

In assumption 1, the code i of an agent is identified with a pure strategy e^i in a game. The notation should not distract the reader, i could have been used instead of e^i. Identification of a code with a strategy is familiar from evolutionary genetics Smith (1998). During replication of individuals in a population, information is transmitted via DNA. It is straightforward to identify the code of an agent with DNA.

Assumption 1, introducing a payoff to software agents, is part of the modelling of software agents as economic agents. Economic principles have been used before in distributed problem solving Huberman (1988), but in De Wilde et al. (1999) the author has made a start with the analysis of general software agents as economic agents. This paper is part of that project.

The replicator dynamics (2) describe asexual reproduction. Agents do sometimes result out of the combination of code from "parent" agents, but such innovative combinations do not occur very often. On a timescale of one year, the replication of agents will be far more important than the reproduction via combination of parent programs.

In addition to DNA exchange, our species also passes information between individuals via cultural inheritance. This tends to result in behaviour that is a close copy to the behaviour of the "cultural" parent. If agents are to represent humans in an evolving society, they will also exhibit cultural inheritance or social learning, which follows assumption 2 Fudenberg and Levine (1998).

In biological systems, one can distinguish long term macroevolution Stanley (1979), and shorter term microevolution Smith (1998). Assumption 2 can be situated in the field of microevolution. On an even shorter timescale, psychologists observe reinforcement learning. Although the results of reinforcement learning are not passed on to offspring (central dogma of neo-Darwinism), it is possible to cast this learning as replicator dynamics Borgers and Sarin (1997). This adds to the credibility of assumption 2, because software agents will often use reinforcement learning together with replication of code De Wilde (1999).

Biological organisms as well as software agents live in an environment. This is actually the same environment, because software agents act for humans, who live in the biological environment. In the model (2), the change of the environment will be reflected in the change of the payoff function $u(e^i, \mathbf{x})$, which has to be written $u(e^i, \mathbf{x}(t), t)$ to make the time dependence explicit. It is very important to be able to model this change in environment, because a successful strategy or agent type should be as robust as possible against changes in environment.

Another type of evolution that software agents have in common with biological agents is mutation. The notion of evolutionary stable strategy Smith (1998) expresses robustness against mutations. However, mutations can positively contribute to the evolution of a system. Current models tend to concentrate on stochastically perturbing the choice of strategies used Fudenberg and Levine (1998); Cavagna et al. (1999), rather than the random creation of new strategies. Much work still needs to be done in this area.

2 The burden of maintaining diversity

Software has a lifetime, so have agents, and so have living organisms. Unsuccessful strategies will die out. This is an essential part of the model (2). Recently there has been much interest in biodiversity Purvis and Hector (20) and sustainable development Pezzey (1989). As in all population dynamics, one can ask the question whether it is worth artificially maintaining strategies (or agent types) with a low payoff. The research on biodiversity suggests that this is worthwhile indeed.

An agent type is a number $i \in K = \{1, \ldots, n\}$. Assume there is a set K_e of endangered agent types, $K_e = \{i \in K | u(e^i, \mathbf{x}) < a\}$, who have a payoff lower than a. The set K_e will change in time, but the threshold a is fixed.

Assume that a is the minimum payoff required to sustain a viable population. It is now possible to redistribute the payoff between the non-endangered strategies outside K_e, and the endangered ones in K_e in the following way

$$
\begin{aligned}
u(e^i,\mathbf{x}) &= a, \quad i \in K_e, \\
u(e^i,\mathbf{x}) &= u(e^i,\mathbf{x}) - \frac{\sum_{j\in K_e}[a-u(e^j,\mathbf{x})]}{q-|K_e|}, \\
& \qquad i \notin K_e. \qquad (3)
\end{aligned}
$$

This transformation conserves the total payoff

$$\sum_{i\in K} u(e^i,\mathbf{x}).$$

Abbreviate

$$b = \frac{\sum_{j\in K_e}[a-u(e^j,\mathbf{x})]}{q-|K_e|} \qquad (4)$$

To indicate that the payoffs have been changed, we will use q instead of p for the population, and y instead of x for the frequencies.

To derive the differential equations in the state variables y_i, start from (1),

$$q(t)y_i(t) = q_i(t), \qquad (5)$$

take the time derivative of left and right hand side to obtain

$$q\dot{y}_i = \dot{q}_i - \dot{q}y_i. \qquad (6)$$

Using (3), we obtain, for $i \in K_e$,

$$
\begin{aligned}
q\dot{y}_i &= q_i a - \sum_{j\in K} \dot{q}_j y_i, \\
&= q_i a - \sum_{j\in K_e} q_j a y_i \\
&\quad - \sum_{j\notin K_e} q_j[u(e^j,\mathbf{y})-b]y_i, \qquad (7)
\end{aligned}
$$

or

$$\dot{y}_i = y_i\left\{a - \sum_{j\in K_e} y_j a - \sum_{j\notin K_e} y_j[u(e^j,\mathbf{y})-b]\right\}. \qquad (8)$$

Similarly, for $i \notin K_e$, we find

$$
\begin{aligned}
\dot{y}_i &= y_i\{u(e^i,\mathbf{y}) - b - \sum_{j\in K_e} y_j a \\
&\quad - \sum_{j\notin K_e} y_j[u(e^j,\mathbf{y})-b]\}. \qquad (9)
\end{aligned}
$$

We can now simplify this using

$$
\begin{aligned}
& -\sum_{j\in K_e} y_j a + \sum_{j\notin K_e} y_j b \\
&= -\frac{|K_e|}{q}a + \frac{q-|K_e|}{q}\frac{\sum_{l\in K_e}[a-u(e^l,\mathbf{x})]}{q-|K_e|}, \\
&= -\frac{1}{q}\sum_{l\in K_e} u(e^l,\mathbf{y}). \qquad (10)
\end{aligned}
$$

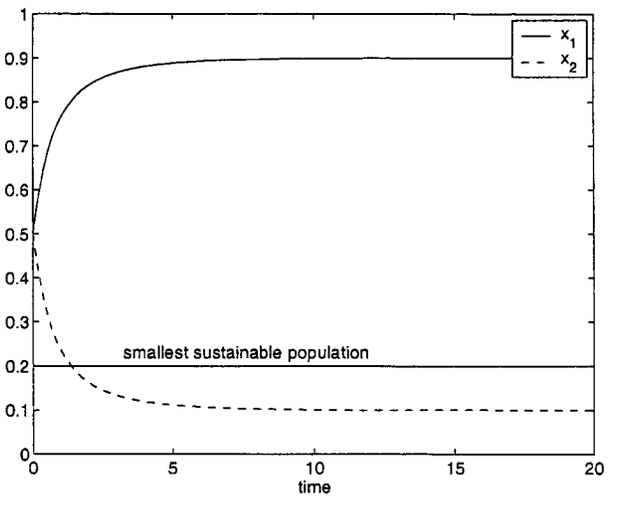


Figure 1: The evolution of the proportion of two population types, x_1 and x_2, for payoffs $u(e^1,\mathbf{x}) = 5x_2$ and $u(e^2,\mathbf{x}) = 0.5$. This is a situation where the first type depends on the second type to survive. Both types survive, but x_2 is low and can become accidentally extinct.

This quantity will be much smaller than both a and $u(e^i,\mathbf{y}), i \notin K_e$, the payoff of the non-endangered strategies, if $u(e^i,\mathbf{y}) \ll a, i \in K_e$, or if $|K_e| \ll q$. These plausible assumptions mean, in words, that the conservation value, a, of the payoff is much larger than the payoff of the endangered strategies, or that there are only a small number of endangered strategies. We will give practical examples in the next section.

Hence we can write the population dynamics with conservation of endangered strategies as

$$
\begin{aligned}
\dot{y}_i &= y_i[a - \sum_{j\notin K_e} y_j u(e^j,\mathbf{y})], \quad i \in K_e, \\
\dot{y}_i &= y_i[u(e^i,\mathbf{y}) - b - \sum_{j\notin K_e} y_j u(e^j,\mathbf{y})], \\
& \qquad i \notin K_e. \qquad (11)
\end{aligned}
$$

Compare this to the population dynamics without conservation Weibull (1995),

$$\dot{x}_i = x_i[u(e^i,\mathbf{x}) - \sum_{j\in K} x_j u(e^j,\mathbf{x})], \quad i \in K. \qquad (12)$$

The term subtracted from the payoff for strategy i is the population average payoff

$$u(\mathbf{x},\mathbf{x}) = \sum_{j\in K} x_j u(e^j,\mathbf{x}). \qquad (13)$$

These effects of artificially increasing the utility for endangered species are illustrated in figures 1 and 2.

3 Niches and Islands

Comparing equations (11) and (12) can teach us a lot about the cost and implications of conservation. In the

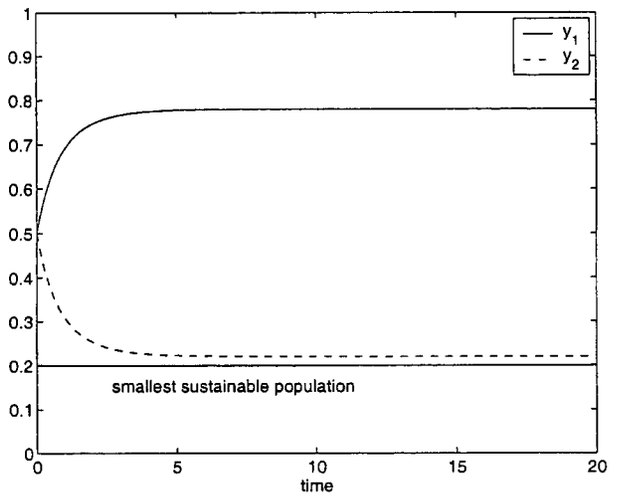


Figure 2: The evolution of the proportion of two population types, y_1 and y_2, as in figure 1, but the payoffs have now been adjusted according to (3), with $K_e = \{2\}$, and $a = 0.8$. This implies $u(e^1, \mathbf{y}) = 5y_2 - 0.3$ and $u(e^2, \mathbf{y}) = 0.8$. The proportion of type 2 is now higher, and not in danger of extinction. The price to pay is that the proportion of type 1 is now lower.

natural world, endangered strategies tend to be come extinct, unless a niche can be found for them. The niche means that the equations (13) are uncoupled. In that case, the payoff does not depend on the whole state $\mathbf{x}$ anymore, but on a projection of $\mathbf{x}$ on a subspace of the state space.

The niche prevents strategies from going extinct because it imposes a particular structure on the payoff function u. For a fixed u, there is no particular advantage or disadvantage in the existence of a niche, the replicator dynamics go their way, and that is all. However, the environment can change, and this will induce a change in u, the function modeling the payoff or success of strategies. Certain feedback loops in the dynamics can now become active.

Assume a system with two strategies that each operate in their niche

$$\begin{aligned} \dot{x_1} &= x_1[u(e^1, x_1, \mathbf{z}) - x_1 u(e^1, x_1, \mathbf{z}) \\ &\quad -x_2 u(e^2, x_2, \mathbf{z})], \\ \dot{x_2} &= x_2[u(e^2, x_2, \mathbf{z}) - x_1 u(e^1, x_1, \mathbf{z}) \\ &\quad -x_2 u(e^2, x_2, \mathbf{z})]. \end{aligned} \tag{14}$$

Strategy 1 does not have to compete with strategy 2 because $u(e^1, x_1, \mathbf{z})$ is independent of x_2. Similarly, Strategy 2 does not have to compete with strategy 1 because $u(e^2, x_2, \mathbf{z})$ is independent of x_1. The frequencies x_1, x_2 of the strategies remain non-zero. The frequencies of the other strategies $x_3, \ldots, x_n$ are grouped in the partial state vector $\mathbf{z}$. If the function u changes, the dynamics of the frequencies of strategies 1 and 2 will now in general be

$$\begin{aligned} \dot{x_1} &= x_1[u(e^1, x_1, x_2, \mathbf{z}) - x_1 u(e^1, x_1, x_2, \mathbf{z}) \\ &\quad -x_2 u(e^2, x_1, x_2, \mathbf{z})], \\ \dot{x_2} &= x_2[u(e^2, x_1, x_2, \mathbf{z}) - x_1 u(e^1, x_1, x_2, \mathbf{z}) \\ &\quad -x_2 u(e^2, x_1, x_2, \mathbf{z})]. \end{aligned} \tag{15}$$

It is now possible that there is a positive feedback that causes x_1 and x_2 to increase over time. This positive feedback is not guaranteed, but if one of the strategies had become extinct, then the positive feedback could never have occurred at all when u changed.

Remark that such a positive feedback was already possible in (14), because both equations are coupled via the partial state vector $\mathbf{z}$. We are not concerned with this process here. More complex mechanisms of the benefits of altruism have been studied in Becker (1976).

So far the pedestrian justification of conservation: once a strategy is extinct, you cannot benefit from it anymore in the future, if the environment changes. In mathematical terms, once the state space is reduced, many trajectories are excluded, and some could benefit your strategy if the environment changes.

Since Darwin's time, it is known that local optima in a population distribution $\mathbf{x}$ can exist on islands. And recently, we have seen how ending the isolation of islands can destroy the local optimum by the invasion of more successful species. Should we isolate information systems and software agents, so that new types can develop? In that case the replicator dynamics (12) will be altered again. For the evolution on an island, it appears that all species not present on the island are extinct. Call K_r the set of strategies represented on the island. Then the population dynamics is

$$\dot{x_i} = x_i[u(e^i, \mathbf{r}) - u(\mathbf{r}, \mathbf{r})], \quad i \in K_r, \tag{16}$$

where $\mathbf{r}$ is the state vector with zeroes for the strategies not in K_r. When the isolation is ended, these zeroes become non-zero, and we obtain the dynamics

$$\dot{x_i} = x_i[u(e^i, \mathbf{x}) - u(\mathbf{x}, \mathbf{x})], \quad i \in K_r \tag{17}$$

for the strategies on the island. This is illustrated in figure 3. The dynamics (16) and (17) are just the general formulations for the two-strategy niche dynamics described by (14) and (15).

4 Learning by individual agents

The dynamics of software agents differs in another aspect from that of biological systems. Learned behaviour can be passed on to offspring. Agents can be duplicated, retaining all that was learned, e.g. via a neural network De Wilde (1999). The replicator dynamic has to take learning into account. If learning is successful, the payoff for states encountered earlier will increase. If the learning also has a generalization capacity, as happens for neural networks De Wilde (1997), then the payoff for states similar to those encountered earlier will also increase. The

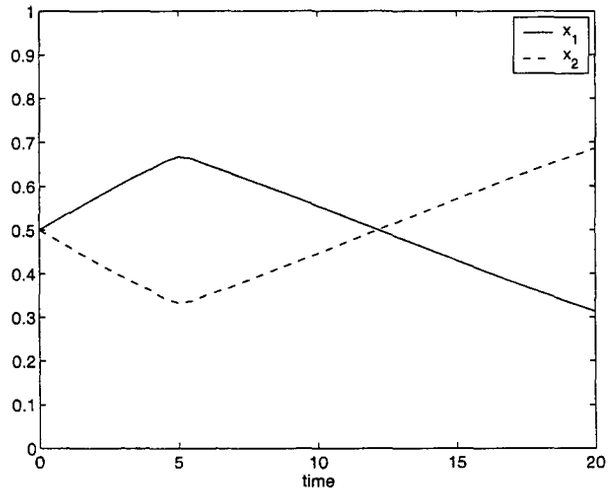


Figure 3: The evolution of the proportion of two population types, x_1 and x_2, with payoffs $u(e^1, \mathbf{x}) = x_1 x_2$ and $u(e^2, \mathbf{x}) = 0.1$. At $t = 5$, the populations become separated, and the positive feedback maintaining type 1 disappears.

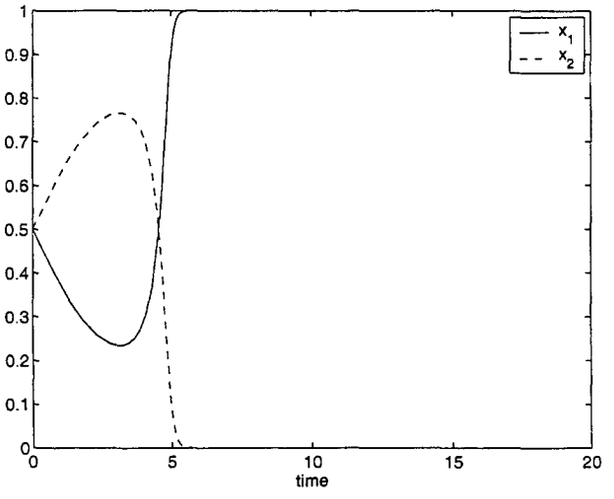


Figure 4: The evolution of the proportion of two population types, x_1 and x_2. Initially type 1 looses out, x_1 goes down. However, type 1 adapts its utility in time as $u(e^1, \mathbf{x}) = t^2 x_1/3$, and this allows it to win over type 2, that uses no learning, $u(e^2, \mathbf{x}) = x_2$.

payoff now changes explicitly with time, and (12) becomes

$$\dot{x_i} = x_i[u(e^i, \mathbf{x}, t) - u(\mathbf{x}, \mathbf{x}, t)], \quad i \in K. \qquad (18)$$

If all the payoffs u were simply multiplied by the same number during learning, the dynamics (18) would be equivalent to (12) in that the orbits remain the same, but they are traversed at a different speed (faster for a constant larger than one). When the payoffs are multiplied by a time-varying factor,

$$\dot{x_i} = x_i \alpha(t)[u(e^i, \mathbf{x}, t) - u(\mathbf{x}, \mathbf{x}, t)], \quad i \in K, \qquad (19)$$

the factor $\alpha(t)$ can be absorbed in the time derivative, and the orbits are traversed with a speed varying in time. When the learning factor $\alpha_i(t)$ becomes dependent on the strategy i however, the orbits are changed, and we cannot compare the population evolution with and without learning any more. A non-trivial learning algorithm for a population of 2 different types is illustrated in figure 4

5 Conclusions

All living things contain a code. So do computer programs, languages, designs and artwork. The code consists of all the information that makes replication possible. In a competitive environment, programs are pitched against each other, in a way similar to individuals in an ecosystem. The interaction brings a payoff u to the program, or language, or design.

The population dynamics with conservation (11) is crucially dependent on the conservation subsidy a per strategy, and on b, which depends on q, the total population, and $|K_e|$, the number of endangered strategies. Conservation maintains a greater pool of strategies than the ecosystem without conservation (12). This makes it possible that the fitness of any single non-endangered strategy could increase when the environment changes adversely for that strategy, via the mutual-benefit feedback loop with an endangered strategy. *The price to pay for this is an overall decrease of the payoff values of the non-endangered strategies.*

In the animal and plant kingdoms, the number of endangered species seems much smaller that the number of non-endangered ones Purvis and Hector (20), although there is great uncertainty on the numerical data. In this situation, (11)-(13) seem to indicate that it is possible to conserve the endangered species, if the effort is spread over all other species. However, replicator dynamics are not such good models of sexual reproduction and mutation, so that it is difficult to reach conclusions.

In the case of languages, artificial and computer, and information systems, the number of endangered types is of the same order as the number of non-endangered ones. In this case, (11)-(15) show that a conservation effort will decrease the payoff of the non-endangered types so much, and their dynamics affected to such an extent, that they also could become extinct if the environment changes.

If conservation is successful, i.e. without lowering too much the payoff for the non-endangered types, it has to go on forever, because the non-endangered types still get a good payoff, so that they continue to thrive, and continue to endanger the endangered types. This is not sustainable if the number of endangered ones is of the same order as the number of non-endangered ones. In other words, *one should not try to control the pure Darwinian evolution in a population of competing agents by artificially maintain-*

ing a diversity of agents.

In short, we have proposed replicator dynamics as a model for the evolution of populations of software agents. We have shown what happens if the utility of some types in increased (conservation), if some types of agents do not interact with each other (niches and islands), and if some types of agents change their utility in time (individual learning). In each of these three cases the adaptation of the population is artificially modified. It is up to the systems analyst to decide which situation applies is a practical case. Our replicator dynamics then allow us to predict what will happen to the different types of agents.

6 Acknowledgements

Thanks to Rachel Yeo for research on software evolution. Partly funded by European Community, under the Future and Emerging Technologies arm of the IST Programme, FET-UIE scheme. This research is linked to, but not part of, the *Evolution and Ecology of Interacting Infohabitants* project.

References

Gary S. Becker. Altruism, egoism, and genetic fitness: Economics and sociobiology. In *The Economic Approach to Human Behavior*, pages 282–294. University of Chicago Press, Chicago, 1976.

T. Borgers and R. Sarin. Learning through reinforcement and replicator dynamics. *Journal of Economic Theory*, 77:1–14, 1997.

Andrea Cavagna, Juan P. Garrahan, Irene Giardina, and David Sherrington. Thermal model for adaptive competition in a market. *Physical Review Letters*, 83:4429–4432, 1999.

Philippe De Wilde. *Neural Network Models, second expanded edition*. Springer Verlag, London, 1997.

Philippe De Wilde. How soft games can be played. In H.-J. Zimmermann, editor, *EUFIT '99, 7th European Congress on Intelligent Techniques & Soft Computing*, pages FSD–6–12698, Aachen, September 1999. Verlag Mainz.

Philippe De Wilde, Hyacinth S. Nwana, and Lyndon C. Lee. Stability, fairness and scalability of multi-agent systems. *International Journal of Knowledge-Based Intelligent Engineering Systems*, 3(2):84–91, 1999.

Drew Fudenberg and David K. Levine. *The Theory of Learning in Games*. MIT Press, Cambridge, Massashusetts, 1998.

B. A. Huberman, editor. *The Ecology of Computation*. North-Holland, Amsterdam, 1988.

Nicholas J. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.

G. Kahen, M. M. Lehman, J. F. Ramil, and P. Wernick. System dynamics modelling of software evolution processes for policy investigation: Approach and example. *Journal of Systems and Software*, 59:271–281, 2001.

John Pezzey. *Economic analysis of sustainable growth and sustainable development*. World Bank, Washington DC, 1989.

Andy Purvis and Andy Hector. Getting the measure of biodiversity. *Nature*, 405:212–219, 20.

John Maynard Smith. *Evolutionary Genetics*. Oxford University Press, Oxford, 1998.

S. M. Stanley. *Macroevolution*. W. H. Freeman, San Fransisco, 1979.

Jörgen W. Weibull. *Evolutionary Game Theory*. MIT Press, Cambridge, Massachusetts, 1995.

Hostile Agents

Robert Ghanea-Hercock
BTexact Future Technologies Group
Adastral Park, Admin 2, pp5, Martlesham Heath,
Ipswich, IP5 3RE, Suffolk, UK.
robert.ghanea-hercock@bt.com

Abstract

"Real stupidity beats artificial intelligence every time."- Bursar 1 - Hex 0 (Terry Pratchett, Hogfather)

There is a co-evolutionary process emerging between hacker techniques and the defence responses of commercial and public networks. Unfortunately the advantage invariably rests with the attackers, as a network is only as strong as the weakest link. In addition recent denial of service attacks make use of large numbers of co-operative soft-bots to launch devastating co-ordinated attacks on target servers, (Dittrich 1999). Increasingly complex and adaptive attack strategies can therefore be expected in the near future. The premise of this paper is that a collective formation of software agents can act as an adaptive and automated defence mechanism for a computer network. Results are presented from a simulation-based model of a co-operative agent security architecture (termed Cosmos), that utilises mechanisms of social-cohesion, trust control and dynamic group formation. The simulation demonstrates how a collaborative defence strategy can successfully eliminate an infectious agent from a host network. In particular the agent community provides a *metabolic sensing array*, which can monitor the macro-scale health of the host network.

1 Introduction

There is an urgent demand within the computing domain for secure systems and networks. Unfortunately the number of successful attacks is increasing via an increasing number of channels, e.g. email, worms, instant messaging, mobile devices, and wireless links. The almost religious belief in rigid corporate firewalls as a perfect defence is finally succumbing to the realisation that no static defence mechanism will ever suffice (Cohen 1997). An example of the issues involved are those associated with current network intrusion detection systems (Balasubramaniyan et al. 1998), i.e.:

- Centrally based data collection and a central analyser system, which is a potential point of failure.
- Limited scalability, due to centralised processing of the monitored systems.
- Slow manual configuration of the analysis process with significant expert intervention required for maintenance and analysis of collected data.
- High operational cost due to the frequent expert intervention required.

1.1 Viral Agents

The difficulty in sustaining a perfect anti-viral defence is equally fraught with difficulty. With the recent advent of powerful macro viruses, (e.g. Melissa and the Love Bug) the limitations of current static or manually controlled security processes is apparent. Unfortunately, current methods of securing computing networks will prove increasingly deficient as the number of individuals engaged in sophisticated attacks increases. Recent reports in the security field have also indicated the key role of internal staff in undermining system security through errors or intentional activity, which are difficult to defend against using firewalls and password measures (Briney 2000). These processes have triggered major research efforts into the use of immune system based models of security (Hofmyer & Forrest 1999 and Kephart 1994). However, such models are difficult to translate into commercially viable systems due to the inadequate metrics available for robustness in complex networks and an insufficient understanding of the key processes in natural immune systems.

1.2 Agent Defence Networks

The agent system presented in this paper is designed to enable the secure transmission of useful pre-processed intrusion data through a large-scale computing network. The system is designed to operate as part of an intrusion detection mechanism, and it may also act as a system for disseminating viral signatures, which have been caught by individual servers. The central proposal is that while the space of all possible attacks is effectively infinite, the space of system responses to those attacks is effectively very small. Within some proposed IDS (Cohen et al 1998), this approach is defined as using "behavioural change detectors". The problem is how to distinguish normal from abnormal behaviour.

A useful summary of the advantages of an autonomous agent based IDS is given by Balasubramaniyan et al. 1998. Additional agent IDS methods can be found in Crosbie & Spafford 1995, who provide a useful definition of the approach:
"- *a group of free-running processes which can act independently of each other and the system. These are termed Autonomous Agents. They are trained to observe system behaviour, and cooperate with each other, so as to flag any behaviour that they consider to be anomalous."*

In order to develop some understanding of the dynamics of agent interactions and group cohesion on the integrity of complex networks we selected a multi-agent model as an experimental platform. Using this tool we investigated a range of behaviours which might influence the robustness and integrity of such a society of interacting agents. Within the agent group an attack on any one agent or host machine should be visible to other agents within the local domain of the agent. Hence warnings and defence solutions can be rapidly propagated throughout the network. The system therefore requires an underlying distributed IDS platform to supply data on intrusions or viral attacks, (currently in development).

1.3 Metabolic Rate Sensors

The key concept is to create an independent measure of the normal operational state of the network to be defended. By comparison with biological agents we need to create a measure of the *metabolic rate* of the services and data flow on the host network. This is achieved through each defence agent monitoring the flow of inter-agent traffic, in addition to monitoring any local host-specific intrusion sensors, such as port scanning alerts. By analogy with natural agents the first stage of most medical tests is to measure a few macro-scale variables, which are strong predicative indicators of the health of the organism, i.e. temperature and blood pressure.

Section 2 presents an overview of co-operative agent systems. Section 3 considers the issues influencing robust and secure behaviour in societies of artificial agents, while section 4 presents results from the agent simulation. Section 5 covers an initial analysis of these results and a summary is offered in section 6.

2 Security Agents

Co-operative software agents have been frequently proposed as a tool for a variety of information processing tasks, i.e. e-commerce transactions (Maes et al. 1999), work-flow modelling or as personal assistants (Etzioni 1996). However, relatively little research has considered their role as an element of network security, exceptions being (Crosbie & Spafford 1995, Helmer et al 1998). Indeed some classes of agents, i.e. Mobile Agents, have been perceived as a serious security threat to any host network, (Chess 1998).

2.1 Social Agent Models

The first question we need to address is what properties of a software agent make them suitable for inclusion in a security system. The following definitions highlight some of the relevant agent attributes:
"*a self-contained program capable of controlling its own decision-making and acting, based on its perception of its environment, in pursuit of one or more objectives.*" (Jennings & Wooldridge 1996.

Software agents therefore possess a number of useful properties that would be beneficial in the construction of a distributed adaptive security system. In particular the ability to sense their environment and take proactive decisions against potential threats. Indeed we may consider current viruses and worms as forms of hostile software agents, as they generally fulfil most of the defining criteria for such agents.

2.2 Group Formation

The particular aspect of multi-agent systems that we have focused on is their ability to form dynamic social groups. The interest in this behaviour stems from the concept that by linking together the sensory and intelligence capabilities of a large number of agents distributed across a network we can *amplify* the ability of the network to resist attacks or intrusion. Specifically through social co-operation, agents can benefit from the combined defensive capabilities of their particular group. In a similar manner natural systems have widely employed distributed sensing and defensive agents, for example within the social insects (Wilson 1971), and within multi-cellular organisms in the form of an immune-system (e.g. Segel 1998). In traditional human military systems the power of a cohesive sub-unit has been well established, for example Roman turtles were a formidable force for several centuries. Modern forces now rely heavily on a cohesive and distributed command and control system; with multiple sensing and intelligence sources.

3 System Design

This project utilises a model of co-operative software agents to investigate how they may contribute to an adaptive computing security architecture. Specifically, a number of software agents are located on host servers within a network and use co-operative behaviours in order to build a distributed intrusion knowledge base. The agents utilise the following methods in order to monitor for network intrusions:

a) Local observation of known system weaknesses, e.g. port attacks, firewall attacks, denial of service attacks and fake user access. (These are prior art methods

required to perform higher level functions by the agents).

b) Each agent maintains a set of internal macro scale variables, which correlate with the state of the network. This is defined as the *metabolic rate* of each agent. This may be defined as an *indirect measurement* of intrusion processes.

Group monitoring. Each agent periodically communicates with a number (N) of other agents within the defined network and exchanges a data set for recently gathered local observations of the network. Hence useful warnings of hostile events are rapidly disseminated throughout the network.

If an agent detects an anomalous event it selects a response based on a previously defined management policy, i.e. attempt to apply corrective measures to halt the event (e.g. close a port on a server) or notify an administrator, via a message channel (email or direct message).

3.1 Prior Art

Using a collective formation of smart software agents to form an adaptive immune-response structure within a network has been discussed in existing literature. Some preliminary work in this field has already demonstrated the effectiveness of such methods (Filman & Linden 1996, and Yialelis, Lupo & Sloman 1996). In particular work by Helmer et al (1998) demonstrates a multi-agent network defence system in which software agents monitor low-level network activity and report it to higher-level software agents for analysis. In the system proposed by Crosbie and Spafford (1995) a similar distributed set of agents monitors network traffic and machine activity, including CPU utilisation. Their system also has agents exchanging anomaly reports between each other and decisions to raise an intrusion alert are based on the combined evidence from a number of agents. This system also utilises a form of machine learning based on evolving Genetic Programs in order to recognise new patterns of attack.

The work by Carver et al (2000) demonstrates the use of a distributed heterogeneous group of agents as an IDS solution. The focus is on dynamic and adaptive response to varying levels of security threats. Work by Balasubramaniyan et al (1998) discusses a detailed design and methodology with common features to the proposed COSMOS system and model.

The work by Qi He and Sycara (1998) demonstrates the use of encrypted KQML message exchange among a networked group of agents which is used for secure PKI certificate management. Although, the application is only partly related, the underlying techniques share some features with the proposed Tag-exchange scheme.

3.2 Comparison

The COSMOS project shares the concept of using a distributed set of co-operative software agents and adds the following novel feature:

Metabolic monitoring. In a similar manner to a human doctor measuring a patients pulse and temperature, COSMOS uses macro-scale changes in the behaviour and processes in a network in order to detect anomalous states, corresponding to attacks. The system can then respond to any type of attack whether intrusions or viral attacks, unlike existing IDS systems.

Current IDS systems rely on specific network or machine checkpoints to indicate the existence of an attack. This requires a separate specific sensing capability for every possible point of attack. While the COSMOS agents may monitor a number of known entry points to the network, they also monitor a set of internal (to the agent) and network state variables. Examples being CPU load, traffic rates and service levels in order to calculate the current metabolic rate of the system and each agent. The COSMOS system can therefore generate warnings due to novel and previously unknown intrusions.

3.3 Group Formation

A particular aspect of multi-agent systems that we have focused on is their ability to form dynamic social groups. The interest in this behaviour stems from the concept that by linking together the sensory and intelligence capabilities of a large number of agents distributed across a network we can amplify the ability of the network to resist attacks or intrusion. Specifically through social co-operation, agents can benefit from the *combined defensive capabilities* of their particular group. In a similar manner natural systems have widely employed distributed sensing and defensive agents in the social insects (Wilson 1971) and within multi-cellular organisms in the form of an immune-system (e.g. Segel 1998).

4 Results – The Cosmos Model

4.1 Experiments

The agent simulation was developed using the REPAST agent toolkit from the University of Chicago (http://repast.sourceforge.net/). This package was selected in preference to alternative social agent platforms e.g. Swarm (Burkhart & Burkhart 1994), as it offers a fast pure Java implementation with extensive support for visualisation, offline batch running and object management.

We first constructed a two-dimensional discrete spatial world model, shown in figure 2, in which a population of

artificial agents could interact and move, based on the Sugarscape model (Epstein and Axtell 1996). This model was selected as it represents a suitable test case environment for investigating complex multi-agent simulations. In addition we wished to enable easy reproduction of the presented results.

4.2 Model Description

The model is based on a population of agents, which are initialised randomly with the following set of variables:

i) **Vision** – an agent can sense other agents and food objects within a specified radius from its own co-ordinates. Assigned randomly within a specified range e.g. 1-5 steps.

ii) **Metabolism** – agents have an integer counter which represents their rate of energy consumption. Assigned randomly in a specified range. Increased whenever an agent is infected with a pathogen.

iii) **Lifespan** – agents are initialised with a fixed lifespan, randomly assigned, typically between 20 – 200 time steps.

iv) **Sugar** – agents require sugar to survive, which is an environmental resource. Sugar re-grows once consumed by an agent at some specified rate. Agents consume sugar by decrementing the value proportional to their metabolic rate. This would translate into an agents consumption of local CPU and machine resources.

v) **Spice** – as described in the Epstein & Axtell model, a second commodity was introduced into the world which is only available from other agents, and is required for agent survival. Agents can only acquire spice when they engage in a trade interaction with another agent. The rules of trade are described in the following section.

vi) **Immune system** – agents have an array of N characters, which represents a simplified immune system.

vii) **Pathogens** – agents may be initialised with a dynamic array of viral infections, composed of short random character strings. Section 4.2.c describes the infection process.

The simulation uses a tagging scheme on each agent in order to distinguish separate social groups of agents. The purpose of this design is to enable multiple groups of agents to coexist within the same physical Intranet environment and maintain independent operations and behaviours. In a physical realisation of the model we envisage the tags representing IP sub-domains.

Agent Classes – there are ten separate social classes defined in the model, based on an array of identifying cultural tags, (the classic model typically only uses two agent classes, Epstein and Axtell 1996). The cultural tags are represented by an array of integer values between 0-9. The specific tags used by the agents were:

Tag 1 is a group identifier, i.e. this tag defined the class to which an agent belongs.

Tag 2 defines how much spice an agent will share during trade.

Tag 3 has no direct role but is used to help measure cultural variance, during agent trade interactions. A visual interface colour codes agents according to the value of tag 1.

Cultural Variance – this model measures variance between the agents by the summed difference between the tag values, in contrast the SugarSpace model uses binary tags and a Hamming distance measurement of cultural or social variance. An integer representation was selected as this maps into the large number of agent classes being operated on.

4.3 Experimental Objectives

4.3.1 Dynamic Group Formation

The first experiments were designed to study under what conditions socially co-operative groups of agents would spontaneously develop, using the defined model. Related work on the dynamic formation of social groups has been shown in work by Hales (Hales 1998); using a similar memetic tag exchange mechanism.

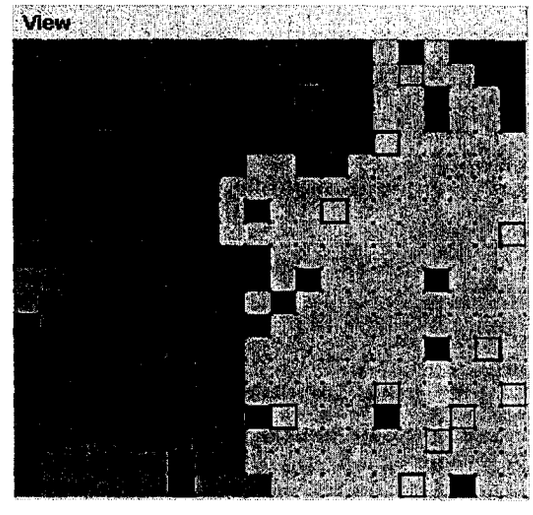


Figure 1. Screen capture of the agent spatial environment, showing 400 agents in a 20x20 grid space environment, in which two dominant social groups have formed. Highlighted cells represent agents, which are currently infected with at least one disease vector.

4.3.2 Rules for Trading Interactions

An agent receives an amount of spice equal to the second tag value T of the agent it is trading with. Each agent

applied the following algorithm during its allocated time slot.

Look in all neighbouring (von Neuman) cells to radius = vision parameter.
If cell occupied then
    If agent is of similar class type then
        trade with agent in the cell
        randomly flip one tag of agent to match own tag
        infect agent with N viral strings
        receive N antibody strings from the agent
Else if cell unoccupied record sugar present.

4.3.3 Immune System Development

The second stage of the work involved adding an artificial immune model to the agents. During each trade interaction between two agents, the initiating agent is passed a vector of N disease strings. Each string is a short sequence of characters, which the receiving agent then attempts to find a match for from its internal immune system, (a large array of character strings). Each string which the agents fails to match results in an increment to its metabolic rate. This results in a gradual degradation of an agent's ability to survive and a reduction in its lifespan. The agents can also undergo random mutation of elements in their immune system, in order to generate potentially novel "anti-body" solutions to current diseases in the population.

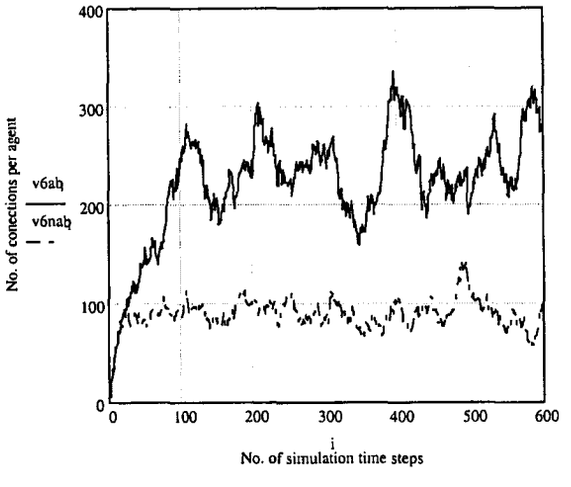


Figure 2. Graph of average number of connections per agent with 6 disease strings per agent. Upper trace shows the effect on the average health of the agents of allowing a co-operative exchange of anti-body vectors between agents during trading interactions.

A second process was available in the simulation to allow agents to exchange a copy of the anti-bodies they have acquired, to each agent they trade with. Figure 4 illustrates the impact of allowing this co-operative inter-agent exchange to occur. The average number of social connections in the population more than doubles, indicating a significant increase in the agents state of health. This is also reflected in greater stability and lifespan of their social groups.

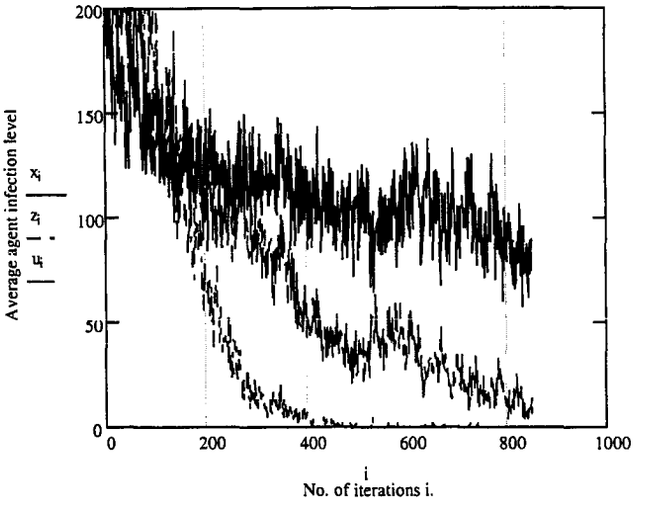


Figure 3. Graph showing decrease in average infection level with shared antibody process enabled between agents, (curve x_i trust level = 1 (low), u_i trust level = 4 (medium), z_i trust level = 9 (high)).

By sharing useful solutions to infections the agent population is able to eliminate the majority of infections in the case of a high degree of trust between the agents. The residual level is due to new agents joining the network and introducing new infections.

5 Analysis

From the above results we extracted the following conclusions.

i) Through a virtual commercial trade process we created stable self-organising groups of agents, via inter-agent meme transfer.
ii) The groups demonstrated resilience to invasion and competition by competing groups.
iii) The groups displayed a collective immunity mechanism, allowing them to withstand frequent infection from external or internal agents.

The metabolic conversions of such a cluster/group therefore contribute to defining its sense of self, (i.e. ability to recognise self-elements). Hence abnormal perturbations of the metabolic rate may be one method for agents to detect when attacks or intrusions are in progress.

5.1 Collective Security

We are currently implementing a multi-agent IDS system across our local Intranet environment with the aim of

detecting hostile behaviour and attacks in the network. This prototype is based on the FIPA JADE agent platform, which provides core messaging and agent visualisation services. In order to transport the artificial antibody signatures between the agents we have designed an encrypted XML formated ACL message structure. The agents will use an encrypted tag exchange scheme to establish group identity and exchange the encrypted objects containing local data representing antibodies to any anomalous behaviour. The agents currently use a number of autonomous http servers to communicate the antibody messages in a secure and asynchronous manner, (all messages are stored in a file database and processed by a separate router service). The proposed method is to install a software agent in every node/server within a network, which can communicate over secure channels with other agents in the local group. Each node or server agent then broadcasts hashed encoded objects (crudely equivalent to T-cells) into the local cluster or group (cellular sub-domain), transmitting its current state and identity. All members of the related cluster listen to this exchange and perform pattern recognition of the messages. Hence, an attack on any individual node or group triggers a collective response. The result is an embedded adaptive and tactical defensive capability within a computer network.

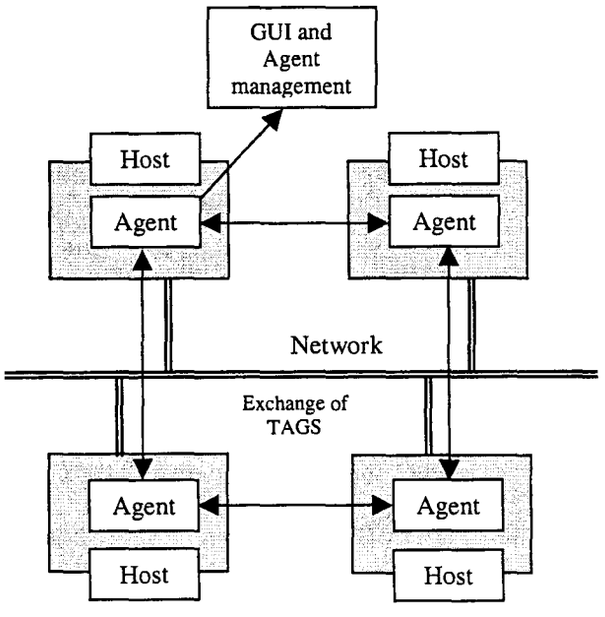


Figure 4. Overview architecture of the proposed agent IDS implementation.

In figure 4 a simplified view of the proposed co-operative agent IDS is illustrated. The system will include a facility to allow administrators to interact with the agents via a GUI and agent management tool. These tools will generally be centrally operated within the network, but may be initiated via a web browser interface from any authorised server, hence preserving the fully distributed advantages of the agent IDS.

The agent group once formed uses the exchange of software antibodies, i.e. Tags (data-sets corresponding to hostile events), in order to develop a *group immunity* to hostile attacks. The following is a list of the main advantages provided by the antibody model.

a) The tag exchange scheme is independent of any specific learning mechanism the agents are using, hence different adaptive strategies and processes can be implemented within this framework.
b) The messaging overhead is very low, as each antibody tag is a short sequence of bytes (typically less than 500 bytes)
c) The mechanism is very robust, i.e. if individual agents are damaged the rest of the group continues functioning.
d) The system is self-organising, the full implementation of the system will incorporate a distributed directory service, such that agents can locate other active agents and connect to them.
e) By specifying new tag identifying elements an administrator can select for a number of sub-groups or cells to form within a network. This increases the robustness and security of the system as each group will be self-maintaining.

A specific application domain for the project is in defending peer to peer (P2P) networks. In particular P2P class networks are currently difficult to secure using existing network security methods as they bypass traditional firewall mechanisms, and may span multiple corporate networks. The proposed antibody exchange and agent processes would easily operate over such P2P networks and enable a higher degree of security for any data or services.

6 FUTURE WORK

Clearly the immune system model used in this agent simulation is an extreme simplification of biological immune mechanisms. Further work will extend the agent immunity model to incorporate more specific processes from the biological domain, such as the use of T cell type agents to recognise when a normal agent has been infected by a malicious process.There are several channels for further development of this work, which need consideration. Firstly, it may be beneficial to allow specialisation by role of the agents, such that agents can become expert at particular aspects of system security, e.g. virus detection. In order to do so an evolution mechanism may be incorporated such that agents are able to generate new agents via genetic recombination, which would be selected for against specific security requirements.

A second level of protection can be incorporated by dividing the network into virtual sub-domains or cells, (i.e. local

virtual private networks with a peer to peer messaging structure). Each cell will contain a separate population of agents with its own unique inter-agent tag sequence. Hence even if an attack succeeds in penetrating one of the agent communities and subverts the agents in that group, it will still have to penetrate the remaining cells individually. In addition if one cell is infected the neighbouring cells would become alerted by changes in the behaviour of that cell.

Open issues include what learning mechanisms the agents should employ to integrate this method with existing anti-viral solutions and what percentage of system resources the agent mechanism will consume.

An interesting theoretical point, which also requires further investigation, is what impact the underlying network topology may have on the infection rates of computer viruses. Future work will incorporate alternative connection topologies between the agents i.e. scale-free and small world designs. Recent work has demonstrated that such topologies can have significant effects on the spreading rate of viral infections, (Pastor-Satorras & Vespiganni 2001). In particular if an anti-viral system can target the most infected nodes in a network and isolate these then the rate of infection can be greatly reduced. Related studies on the robustness of complex networks during malicious attacks has generated predictive equations for network attack tolerance, (Saffre & Ghanea-Hercock 2001).

7 Conclusions

The global computing and communications network is a rapidly growing adaptive and dynamic structure on an immense scale. Future attempts to defend Intranet or telecommunication networks will require equally adaptive and increasingly automated processes. This work indicates that a cohesive network of socially interacting agents can create a highly robust and adaptive defence system for information networks. The agent simulation we have developed demonstrates that it is possible to create a population of autonomous agents, which form self-healing social groups with greater resistance to attacks and perturbation than isolated agents. A key parameter of such co-operation is the degree of trust which is established between agents within the same domain, as increased levels of trust can assist in the rapid diffusion of anti-body solutions. (Although at the risk of corrupted agents exploiting such trust).

The process of continuous inter-agent meme transfer enables the agents to maintain a measure of group identity, which is essential to the process of distinguishing self from non-self. In addition the cooperative exchange of recognised patterns for hostile pathogens/viruses greatly improves the immune response of such an agent community.

References

1. Albert R., Jeong K., and Barabási A., "Attack and error tolerance of complex networks", Nature 406 378 (2000).
2. Anderson R., Feldman P., Gerwehr S., Houghton B., Mesic R., Pinder J, Rothenberg J., and Chiesa J. (1999). "Securing the U.S. Defense Information Infrastructure: A Proposed Approach." MR-993-OSD/NSA/DARPA. www.rand.org/publications/electronic/info.html
3. Balasubramaniyan J., Jose Omar Garcia-Fernandez, Spafford E., and Zamboni D. "An Architecture for Intrusion Detection using Autonomous Agents". Department of Computer Sciences, Purdue University; Coast TR 98-05; 1998.
4. Briney A., "Security Focused", Online report on Information system security, from http://www.infosecuritymag.com 2000.
5. Burkhart B., Burkhart R., "The Swarm Multi-Agent Simulation System"(OOPSLA) '94 Workshop on "The Object Engine" 7 September 1994.
6. Carver C.A., Hill J.M, Surdu J.R., and Pooch U.W., "A Methodology for using Intelligent Agents to provide Automated Intrusion Response," IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop, West Point, NY, June 6-7 2000, pp. 110-116.
7. Chess D.M. (1998). "Security Issues in Mobile Code Systems", pp.1-14. In: Mobile Agents and Security, ed. G. Vigna. Springer-Verlag.
8. Cohen F., " 50 Ways to Defeat your Intrusion Detection System", onlien paper at http://all.net/journal/netsec/1997-12.html.
9. Crosbie M. and Spafford E. "Defending a Computer System using Autonomous Agents", In 18th National Information Systems Security Conference, oct 1995. http://www.cs.purdue.edu/homes/mcrosbie/research/NISSC95.ps.
10. Dittrich D., " The "Tribe Flood Network" distributed denial of service attack tool ", online report at http://staff.washington.edu/dittrich/misc/tfn.analysis.txt.
11. Epstein J., Axtell R., "Growing Artificial Societies: Social Science from the Bottom Up", MIT Press, 1996.
12. Etzioni O., "Moving up the information food chain: Deploying softbots on the world-wide web". In Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI96) , Portland, OR, 1996.
13. Filman R., and Linden T., "Communicating Security Agents", Proc. WET ICE 1996.
14. Forrest S., Perelson S., Allen L., and Cherukuri R., "Self-Nonself Discrimination in a Computer". In Proceedings of IEEE Symposium on Research in Security and Privacy, pages 202-- 212, Oakland, CA, 16-18 May 1994.

15. Helmer G.G., Wong J.S., Honavar V., and Miller L. "Intelligent agents for intrusion detection". In Proceedings, IEEE Information Technology Conference, pages 121--124, Syracuse, NY, September 1998.
16. Hofmeyr S., & Forrest S. "Immunity by Design: An Artificial Immune System" Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Morgan-Kaufmann, San Francisco, CA, pp. 1289-1296 (1999).
17. Kephart J.O., " A Biologically Inspired Immune System for Computers", Artificial Life IV, Proceedings of the Fourth International Workshop on Synthesis and Simulatoin of Living Systems, Rodney A. Brooks and Pattie Maes, eds., MIT Press, Cambridge, Massachusetts, 1994, pp. 130-139
18. Maes P., Guttman R. and Moukas A., "Agents that Buy and Sell: Transforming Commerce as we Know It." Communications of the ACM, Mar 1999 Issue, available online from http://ecommerce.media.mit.edu /Kasbah/.
19. MAFTIA "Malicious and Accidental Fault Tolerance for Internet Applications", IST Programme RTD Research project, 2001, http://www.newcastle.research.ec.org/maftia/summary.html
20. Moody J.& White R.D.,"Social Cohesion and Embeddedness: A Hierarchical Conception of Social Groups", Submitted to American Journal of Sociology 2000.
21. Pastor-Satorras, R. & Vespignani, A. "Epidemic spreading in scale-free networks". Physical Review Letters, 86, 3200 - 3203 (2001).
22. Saffre F. and Ghanea-Hercock R., "Simple Laws for Complex Networks", International Conference on Dynamical Networks in Complex Systems, Kiel, July 2001.
23. Segel, A., and R. Lev Bar-Or. "Immunology Viewed as the Study of an Autonomous Decentralized System." In Articial Immune Systems and Their Applications, edited by D. Dasgupta, 65-88. Berlin: SpringerVerlag, 1998.
24. Watts, D. & Strogatz S. "Collective Dynamics of 'small-world' networks", Nature 393, 440-442 (1998).
25. Wilikens M., Jackson T. "Survivability of Networked Information Systems and Infrastructures", EU DG III/F, Deliverable report on Survivability, Joint research Centre Italy, 1997.
26. Wilson, E.O. (1971) The insect societies. Harvard University Press.

Improving on the reinforcement learning of coordination in cooperative multi-agent systems

Spiros Kapetanakis and Daniel Kudenko
Department of Computer Science
University of York, Heslington
York, YO10 5DD, U.K.
{spiros, kudenko}@cs.york.ac.uk

Abstract

We report on an investigation of reinforcement learning techniques for the learning of coordination in cooperative multi-agent systems. These techniques are variants of Q-learning (Watkins, 1989) that are applicable to scenarios where mutual observation of actions is not possible. To date, reinforcement learning approaches for such *independent* agents did not guarantee convergence to the optimal joint action in scenarios with high miscoordination costs. We improve on previous results (Claus and Boutilier, 1998) by demonstrating that our extension causes the agents to converge almost always to the optimal joint action even in these difficult cases.

1 Introduction

Learning to coordinate in cooperative multi-agent systems is a central and widely studied problem, see, for example Lauer and Riedmiller (2000); Boutilier (1999); Claus and Boutilier (1998); Sen and Sekaran (1998); Sen et al. (1994); Weiss (1993). 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.* We investigate the case of *independent* agents that cannot observe one another's actions, which often is a more realistic assumption.

In this investigation, we focus on reinforcement learning, where the agents must learn to coordinate their actions through environmental feedback. To date, reinforcement learning methods for independent agents (Tan, 1993; Claus and Boutilier, 1998; Sen et al., 1994) did not guarantee convergence to the *optimal* joint action in scenarios where miscoordination is associated with high penalties. We investigate variants of Q-learning (Watkins, 1989) in search of improved convergence to the optimal joint action. More specifically, we investigate the effect of the temperature function on Q-learning with the Boltzmann action-selection strategy. We validate our results experimentally and show that the convergence probability is greatly improved over other approaches.

Our paper is structured as follows: we first introduce a common testbed for the study of learning coordination in cooperative multi-agent systems and describe an especially difficult variant with high miscoordination costs. We then introduce variants of Q-learning and discuss the experimental results. We finish with an outlook on future work.

2 Single-stage coordination games

A common testbed for studying the problem of multi-agent coordination is that of repeated cooperative single-stage games (Fudenberg and Levine, 1998). In these games, the agents have common interests i.e. they are rewarded based on their joint action and all agents receive the same reward. In each round of the game, each agent chooses an action. These actions are executed simultaneously and the reward that corresponds to the joint action is broadcast to all agents.

A more formal account of this type of problem was given by Claus and Boutilier (1998). In brief, we assume a group of n agents $\alpha_1, \alpha_2, \ldots, \alpha_n$ each of which have a finite set of *individual actions* A_i which is known as the agent's *action space*. In this game, each agent α_i chooses an individual action $action \in A_i$ from its action space to perform. The action choices make up a *joint action* which is associated with a unique reward. Upon execution of their actions all agents receive the reward that corresponds to the joint action. For example, Table 1 describes the reward function for a simple cooperative single-stage game. If agent 1 executes action b and agent 2 executes action a, the reward they receive is 5. Obviously, the optimal joint-action in this simple game is (b, b) as it is associated with the highest reward of 10.

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

Table 1: A simple cooperative game reward function.

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 is associated with each 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 strategy. In contrast, independent learners must estimate the value of their individual actions based solely on the rewards that they receive for their actions.

A popular technique for learning coordination in cooperative single-stage games is one-step Q-learning which is due to Watkins (1989), 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. These values are updated after each step of the game according to the reward received for the action. The value $Q(action)$ provides an estimate of the usefulness of performing action *action*. We apply Q-learning with the following update function:

$$Q(action) \leftarrow Q(action) + \lambda(r - Q(action))$$

where λ is the learning rate ($0 < \lambda < 1$) and r is the reward that corresponds to choosing action *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, Q-learning will 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. A major challenge in defining a suitable strategy for the selection of actions is to strike a balance between exploring the usefulness of moves that have been attempted only a few times and exploiting those in which the agent's confidence in getting a high reward is relatively strong. This is known as the *exploration/exploitation problem.*

The action selection strategy that we have chosen for our research is the Boltzmann strategy (Kaelbling et al., 1996) which states that agent α_i chooses action *action* with a probability based on its current estimate of the usefulness of that action. In the case of Q-learning, the Q values act as the agent's estimates of the usefulness of an action so the probability for action selection is based on the function:

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

Specifically, we have concentrated on a proper choice for the temperature function T. This function provides an element of randomness in the way that actions are chosen: high values in temperature encourage exploration since small variations in Q values become less important whereas low temperature values encourage exploitation. The value of the temperature can be decreased over time as exploitation takes over from exploration. It has been shown (Singh et al., 2000) that convergence to a joint action can be ensured if the temperature function adheres to certain properties. However, we have found that there is more that can be done to ensure not just convergence to *some* joint action but convergence to the *optimal* joint action, even in the case of independent learners.

In our study, we focus on the *climbing game* which is due to Claus and Boutilier (1998). This focus is without loss of generality since the climbing game is representative of coordination problems with high miscoordination penalty in multi-agent systems and is, therefore, especially difficult to solve. This game is played between two agents. The reward function for this game is included in Table 2:

		Agent 1		
		a	*b*	*c*
Agent 2	*a*	11	-30	0
	b	-30	7	6
	c	0	0	5

Table 2: The climbing game table.

For each agent, it is difficult to converge to the optimal joint action (a, a) because of the negative reward in the case of miscoordination. For example, if agent 1 plays a and agent 2 plays b, then both will receive a negative reward of -30. Incorporating this reward into the learning process can be so detrimental that both agents tend to avoid playing the same 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. The reason is as follows: if agent 1 plays c, then agent 2 can play either b or c to get a positive reward (6 and 5 respectively). Even if agent 2 plays a, the result is not catastrophic since the reward is 0. Similarly, if agent 2 plays c, whatever agent 1 plays, the resulting reward will be at least zero. From this analysis, we can see that the climbing game is a sufficiently complex problem for the study of coordination. It includes heavy miscoordination penalties and "safe" actions that are likely to tempt the agents away from the optimal joint action.

3 Experimental results

This section contains our experimental results. Each subsection describes a variant of Q-learning for coordination games where the agents have no social awareness.

3.1 Exponential temperature

Typically, reinforcement learning experiments that use a temperature function to control how much exploration and exploitation an agent performs during the learning are set up so that the value of the temperature starts from an initial value and decreases over time. Exponential decay in the value of the temperature is a popular choice. This way, the agent learns until the temperature reaches some lower limit. The experiment then finishes and results are collected. The temperature limit is normally set to zero which may cause complications when calculating the action-selection probabilities with the Boltzmann function. To avoid such problems, we have set the temperature limit to 1 in our experiments.

Although reinforcement learning experiments that use an exponentially decaying temperature function are quite common, the effect that the parameters of the temperature function have on the learning have not been explored sufficiently.

In our analysis of exponential temperature functions, we use the following family of functions:

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

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. Varying the parameters allows a detailed specification of the temperature function. A sample of 5 of these choices have been plotted in figure 1.

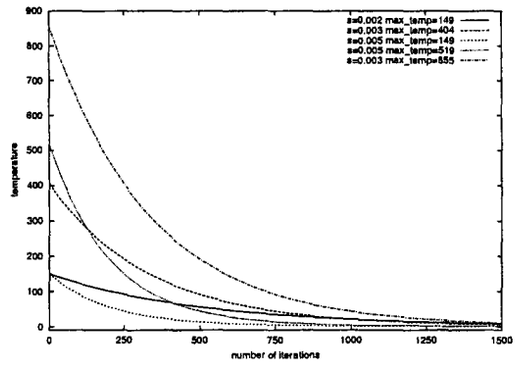


Figure 1: Exponential temperature functions for different s and max_temp values.

For a given number of iterations, we experimented with a variety of s, max_temp combinations. To motivate this point we use the climbing game and set the length of the experiment to 1000 iterations. We repeat each experiment 1000 times to ensure adequate confidence in the results. We compare the following parameter combinations:

- $s = 0.01$ max_temp $= 499$
- $s = 0.006$ max_temp $= 499$
- $s = 0.01$ max_temp $= 999$
- $s = 0.006$ max_temp $= 999$

The results from these experiments are included in Tables 3 to 6. Note that greater values of s mean that the temperature is decaying more rapidly.

	a	b	c
a	193	0	0
b	0	113	275
c	0	0	419

Table 3: Results with $s = 0.01$ max_temp $= 499$.

	a	b	c
a	192	0	0
b	0	114	279
c	0	0	414

Table 4: Results with $s = 0.006$ max_temp $= 499$.

	a	b	c
a	202	0	0
b	0	121	274
c	0	0	403

Table 5: Results with $s = 0.01$ max_temp $= 999$.

	a	b	c
a	154	0	1
b	0	98	331
c	9	0	405

Table 6: Results with $s = 0.006$ max_temp $= 999$.

Tables 3 to 6 contain the number of times out of the 1000 repetitions of the experiment that a joint action was reached after 1000 moves. For example, in table 3, joint action (c, b) was reached 275 times. The success ratio of these experiments is defined as the number of times the agents converged to the optimal joint action (a, a) over the total number of repetitions. These experiments are only a small sample, nevertheless, they show that, for a given length of the experiment, variation in the s and max_temp parameters of the exponential function do not have a significant impact on convergence to the optimal joint action.

3.2 FMQ heuristic

The climbing game is a particularly difficult coordination game. This is not only due to the high miscoordination penalty that is associated with joint actions (a, b) and (b, a). It is also due to the relative safety provided

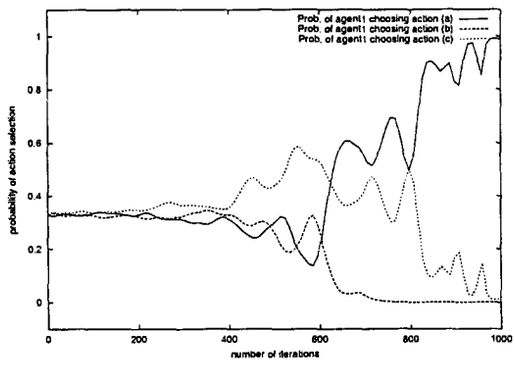


Figure 2: Probabilities for action-selection.

by action c for both agents. Figure 2 depicts the action-selection probabilities for agent1 during a successful run of the experiment for 1000 moves with $s = 0.006$ and $\max_t \text{emp} = 499$.

It is clear from Figure 2 that action c is consistently the most probable action until a dominates it. This happens after approximately 600 moves. In fact, in most unsuccessful runs of the same experiment, action c is the most probable action throughout. What is needed, in this case, is a way to influence the learning towards the optimal joint action. Since independent agents are devoid of social awareness, from the agent perspective, the learning scenario consists only of actions and rewards: each agent knows what actions it plays and what the corresponding rewards are. There are two ways to influence the learning towards the optimal joint action: by changing the Q-update function and by changing the action-selection strategy.

Lauer and Riedmiller (2000) describe an algorithm for multi-agent reinforcement learning which is based on the *optimistic* assumption. In the context of reinforcement learning, this assumption implies that an agent makes any action it finds suitable expecting the other agent to choose accordingly. More specifically, the optimistic assumption affescts the way Q values are updated. Under this assumption, the update rule for playing action α defines that $Q(\alpha)$ is only updated if the new value is greater than the current one. Incorporating the optimistic assumption into Q-learning solves the climbing game every time. This fact is not surprising since the penalties for miscoordination, which make learning optimal actions difficult, are neglected as their incorporation into the learning tends to lower the Q values of the corresponding actions. Such lowering of Q values is not allowed under the optimistic assumption so that all the Q values eventually converge to the maximum reward corresponding to that action for each agent. In this way, using the optimistic assumption solves the climbing game.

The optimistic assumption is a heuristic that applies to the Q update function. Similarly, one can define heuristics that apply to the action-selection strategy. For example, we define the *Frequency Maximum Q value* (FMQ) heuristic. First, we augment the agent's internal status by maintaining 3 values for each of its actions $\alpha \in A_i$:

① action_count(α) holds the number of times the agent has chosen α in the game

② maxR(α) holds the maximum reward encountered so far for choosing action α

③ count_maxR(α) holds the number of times that the maximum reward has been received as a result of playing action α

The expected value function in the Boltzmann strategy is now:

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

where k is a weight which controls the importance of the FMQ heuristic in the action-selection and $\text{freq}(\text{maxR}(\alpha))$ is the frequency of receiving the maximum reward corresponding to an action. $\text{freq}(\text{maxR}(\alpha))$ is defined as:

$$\text{freq}(\text{maxR}(\alpha)) = \frac{\text{count\_maxR}(\alpha)}{\text{action\_count}(\alpha)}$$

Informally, the FMQ heuristic carries the information of how frequently an action produces its maximum corresponding reward. Table 7 contains the results that were obtained using the FMQ heuristic with the climbing game. These results were achieved with an exponentially decaying temperature ($s = 0.006, \text{max\_temp} = 499$) and $k = 10$ over 1000 iterations of the experiment for 1000 moves. Note that the success ratio with the FMQ heuristic is 99.8% whereas the same settings gave a success ratio of only 19.2% with the normal exponentially decaying temperature function (see Table 4).

	a	b	c
a	998	0	0
b	0	2	0
c	0	0	0

Table 7: Results with the FMQ heuristic in the climbing game.

4 Validation

To experimentally validate the FMQ heuristic and compare it to the optimistic assumption Lauer and Riedmiller (2000), we introduce a variant of the climbing game which we term *the partially stochastic climbing game*. This version of the climbing game differs from the original in that one of the joint actions is now associated with a stochastic reward. The reward function for the partially stochastic climbing game is included in Table 8.

Joint action (b, b) yields a reward of 14 or 0 with probability 50%. The partially stochastic climbing game is

		Agent 1		
		a	b	c
Agent 2	a	11	-30	0
	b	-30	14/0	6
	c	0	0	5

Table 8: The partially stochastic climbing game table.

functionally equivalent to the original version. This is because, if the two agents consistently choose their b action, they receive the same overall value of 7 over time as in the original game.

Using the optimistic assumptionLauer and Riedmiller (2000) on the partially stochastic climbing game consistently converges to the suboptimal joint action (b, b). This because the frequency of occurrence of a high reward is not taken into consideration at all. In contrast, the FMQ heuristic shows much more promise in convergence to the optimal joint action. It also compares favourably with normal Q-learning with an exponential temperature function. Tables 9, 10 and 11 contain the results from 1000 experiments with the exponential function, the optimistic assumption and the FMQ heuristic respectively. In all cases, the parameters are: $s = 0.006$, max_temp $= 499$ and, in the case of FMQ, $k = 10$.

	a	b	c
a	212	0	3
b	0	12	289
c	0	0	381

Table 9: Results with exponential temperature.

	a	b	c
a	0	0	0
b	0	1000	0
c	0	0	0

Table 10: Results with optimistic assumption.

	a	b	c
a	988	0	0
b	0	4	0
c	0	7	1

Table 11: Results with the FMQ heuristic.

5 Discussion

The FMQ heuristic performs equally well in the partially stochastic climbing game and the original deterministic climbing game. In contrast, the optimistic assumption only succeeds in solving the deterministic climbing game. However, we have found a variant of the climbing game in which both heuristics perform poorly: *the fully stochastic climbing game*. This game has the characteristic that *all* joint actions is probabilistically linked with two rewards. The average of the two rewards for each action is the same as the original reward from the deterministic version of the climbing game so the two games are functionally equivalent. For the rest of this discussion, we assume a 50% probability. The reward function for the stochastic climbing game is included in Table 12.

		Agent 1		
		a	b	c
Agent 2	a	10/12	5/-65	8/-8
	b	5/-65	14/0	12/0
	c	5/-5	5/-5	10/0

Table 12: The stochastic climbing game table (50%).

It is obvious why the optimistic assumption fails to solve the fully stochastic climbing game. It is for the same reason that it fails with the partially stochastic climbing game. The maximum reward is associated with joint action (b, b) which is a suboptimal action. The FMQ heuristic, although it performs marginally better than normal Q-learning still doesn't provide any substantial success ratios.

6 Summary and Outlook

We have presented an investigation of techniques that can allow two agents that are unable to sense each other's actions to learn coordination in cooperative single-stage games. These technique are applicable to independent learners. However, there is still much to be done towards understanding exactly how the temperature function can influence the learning of optimal joint actions in this type of repeated games. In the future, we plan to specifically investigate the impact of the temperature function parameters on the learning.

Furthermore, since agents typically have a state component associated with them, we plan to investigate how to incorporate such coordination learning mechanisms in multi-stage games. We intend to further analyse the applicability of various reinforcement learning techniques to agents with a substantially greater action space. Finally, we intend to perform a similar systematic examination of the applicability of such techniques to partially observable environments where the rewards are perceived stochastically.

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.

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.

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.

Sandip Sen and Mahendra Sekaran. Individual learning of coordination knowledge. *JETAI*, 10(3):333–356, 1998.

Sandip Sen, Mahendra Sekaran, and John Hale. Learning to coordinate without sharing information. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 426–431, Seattle, WA, 1994.

S. Singh, T. Jaakkola, M. L. Littman, and C Szpesvari. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning Journal*, 38(3):287–308, 2000.

Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.

C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England, 1989.

Gerhard Weiss. Learning to coordinate actions in multi-agent systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, volume 1, pages 311–316. Morgan Kaufmann Publ., 1993.

A Framework for Coherence-Based Multi-Agent Adaptivity

Nick Lacey;Henry Hexmoor
Computer Science and Computer Engineering
University of Arkansas
{nlacey,hexmoor}@uark.edu

Abstract

In this paper we present research which extends previous work concerning the application of philosophical theories to agent knowledge base (AKB) design. We show how the theories and techniques presented in this paper allow multiple agents to adapt to a dynamic environment, and pursue long-term goals and intentions. A system is implemented and described which tests the ability of agents based on the framework to act responsively, pro-actively, and cooperatively in a multi-agent environment.

1 Introduction

A *situated* agent is one which exists within an environment. Such agents receive information about their environment through their sensory equipment, and are able to perform actions in their environment through their effectors. In the case of both human and artificial agents, the world which the agent must interact with will often contain more complexity than can be represented within the internal model of the agent. This means that both human and artificial agents will face problems concerning the accuracy of their model in relation to their environment.

It is possible to form parallels between the problems faced by a situated agent and the questions posed by various areas of philosophy, especially Epistemology, Metaphysics, and the Philosophy of Language. The particular path taken through the philosophical maze in each of these areas has implications for the design and implementation of an artificial agent. As Van Inwagen points out, *everyone* has philosophical beliefs, whether or not they are explicitly aware of them (van Inwagen, 1993). As the designers of most artificial agents are not philosophers, they are implicitly basing their agents on the philosophical assumptions which come bundled with common sense, which Van Inwagen calls the *"Common Western Metaphysic"*.

In (Lacey, 2000), and (Lacey and Lee, 2001), it was shown that it was possible to construct two radically different philosophical positions and then construct artificial agents based on these positions. The performance of the agents was then compared, and it was concluded that the philosophical foundations on which an artificial agent was based do indeed affect its design, implementation, and behaviour.

In this paper we concentrate less on the philosophical motivations for this research, and more on extending the earlier work so as to produce agents which are capable of addressing more complex tasks.

This paper is structured as follows. In Section 2 we briefly describe the design, implementation, and behaviour of the original agent. In Section 3, we describe how the addition of intentions and the ability to represent social concepts to the agent's ontology renders the revised agent, agent **CX**, *pro-active* and able to operate in a multi-agent environment. Section 4 then describes experiments that have been carried out in order to test the new abilities of agent **CX**, the results of which are discussed in Section 5. Finally, Section 7 offers some concluding remarks concerning this work.

2 The Design and Implementation of Agent SH

Agent **SH** was designed to be strongly holistic. This means that the meaning of every belief within its ontology is, as much as possible, determined in relation to *every* other belief. This was implemented using a belief revision mechanism based on explanations, coherence and constraint satisfaction.

Agent **SH**'s low-level perceptions were organised and given meaning using high-level explanations of the agent's environment. The most coherent explanation at any given time was taken to be the correct explanation of the agent's current sensor data.

2.1 Explanations

The primary method of inference used by agent **SH** that of *Explanation-Based Backward Chaining*. The ultimate goal of the backward chaining process is to find the overall explanation which bests explains the current input data. The value of the explanation may be derived from several alternative explanations, each one representing an al-

ternative high-level conception of the current state of the agent's environment.

The first explanation that the agent will attempt to backward chain is the *Default Explanation*. This represents the most coherent explanation of all information received by the agent prior to the present moment. If the agent is able to find the value of the *Default Explanation* without violating any constraints, then it has no need to investigate any of the alternative explanations, and can adopt the default explanation.

Alternatively, the agent may find that no value for default explanation can be found without violating constraints. This indicates the agent's internal model is out of step with its environment, so some adjustment is required. If this occurs, as many alternative explanations as necessary are explored. The alternative explanations are derived and then holistically evaluated by comparing their *integrity* scores. This approach mirrors Thagard and Millgram's concept of the holistic assessment of competing hypotheses, and is based on philosophical approaches to truth and justification based on coherence.[1] (Thagard and Millgram, 1997)

Explanations are *Meta-Level* beliefs, meaning that they concern the relationships between other beliefs, rather than directly representing states of affairs in the agent's environment. Every explanation may incorporate other, lower-level explanations, as well as domain-dependent beliefs concerning facts and relations about the agent's environment. Alternative explanations will usually provide differing mechanisms to derive the same pieces of data. This means that if a piece of sensory data that is necessary for the default explanation is unavailable, the agent may be able to use its knowledge of its environment to derive the missing data from another source.

The precise nature of the explanations used by a given agent will depend upon the environment within which the agent is to operate. Coherence can be used to provide a measure of the "best" explanation. The concepts of *integrity* and *centrality* are used to measure the utility of a possible explanation. Integrity and centrality are used to measure the value of an entire knowledge base, and an individual belief, respectively.

If the ontology of the agent were to be visualised as a sphere, the central beliefs would be the meta-beliefs while the outermost beliefs would be state dependent assertions.[2] By associating each belief with the ontological level to which it belongs, the agent has access to a computationally inexpensive method of deriving the centrality of each belief.

This technique also addresses a common criticism of coherence based justificational structures, namely that they are unable to account for the differing epistemological importance attached to difference classes of belief. The spherical model provides a clear basis for holding that high-level core beliefs are more important to the agent's ontology that perceptual beliefs on the periphery, and hence are less likely to be revised.

[1] For more information on coherence as an approach to truth and justification, see (Audi, 1993) and (Kirkham, 1995).

[2] This structure was inspired by Quine (Quine, 1980).

2.2 Constraints

Constraints are used to alert the agent to the presence of an inconsistency. Their role can be seen as three-fold:

1. Constraints can be viewed as a shortcut, as they allow the agent to detect an inconsistency as early as possible in the knowledge base derivation process.

2. Constraints allow the high-level representation of states of affairs that cannot obtain in the agent's environment. If a particular constraint is violated under the current explanation, the agent can immediately infer that the explanation is flawed, and thus can invest its energies elsewhere.

3. Due to the high-level nature of constraints, it is possible to associate a particular recovery plan with every constraint. By following the recovery plan, the agent may be able to produce a new explanation which successfully resolves the current problem.

Constraints relevant to the agent's domain are provided by the agent designer. As such, constraints represent pre-processed knowledge provided to the agent. Pollock (Pollock, 1998) notes that this technique obviates the agent from having to concern itself with all the intricacies of the domain, and risk encountering the frame problem. While we accept this point, it should be noted that the agent's design is flexible enough that it may be possible to allow it to modify its constraints or add new ones as a result of experience, and indeed this is an interesting area for future developments.

3 Adding Intentions and Social Notions to the Architecture

Agent **SH** was responsive, rather than pro-active. The agent was able to perceive and react to changes in its environment, but was not able to represent or act on long-term plans or goals. Jennings, Sycara and Wooldridge (Jennings et al., 1998) define a responsive system as one which is able to perceive its environment and respond to changes which occur in it.

As such, responsiveness can be contrasted with pro-activity. A pro-active agent does not merely respond to its environment, but is able to exhibit goal-directed behaviour. The approach we used to create a pro-active agent based on agent **SH** was to add *intentions* to the agent's ontology. The BDI approach to agent design recognises the primary importance of beliefs, desires, and intentions (Wooldridge, 2000).

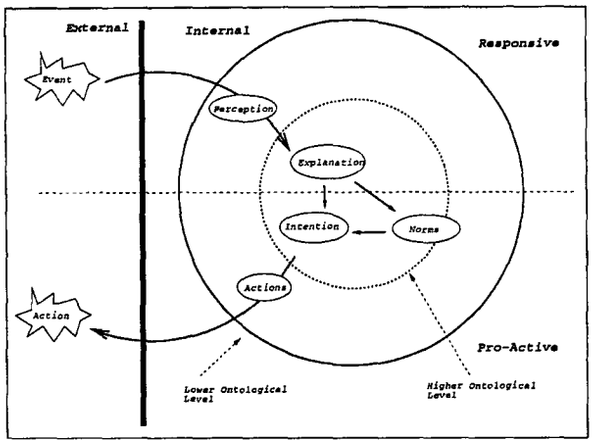


Figure 1: Adding Intentions and Norms to Agent **SH**

Intentions occupy a middle ground between desires and actions which allow the agent to focus on a manageable number of achievable short or long-term goals which will maximise, to the best of the agent's knowledge and abilities, its long-term utility. In other words, intentions allow an agent to be goal-driven, rather than event-driven (Schut and Wooldridge, 2001).

Thus, adding intentions to the architecture in which agent **SH** is based renders the agent pro-active, as well as responsive. The agent created as a result of the extensions to agent **SH** has been called agent **CX**, as it is based on an extended form of coherence.

As shown in Figure 1, the addition of intentions to the architecture of agent **SH** creates a framework based on a two-stage cognitive process. This figure also shows that norms have been added to the agent's ontology. The social aspects of agent **CX** are discussed in Section 3.5. The first stage is responsive, in that the agent uses explanation-based backward chaining to form the most coherent explanation of its current sensor data.

The second stage, however, is pro-active. During this stage, agent **CX** uses the high-level explanations generated during the first stage as a guide when forming an *intention tree*. Intentions relevant to the current domain are combined using backward chaining, in much the same way as explanations were combined during the responsive stage of the agent's cognitive cycle.

3.1 Intentions, Plans, and Actions

An interesting feature of this architecture is that the agent does not represent *plans* explicitly. This is not to suggest that the agent does no planning. Clearly, some form of planning is essential if the agent is to achieve long-term goals. However, in the architecture we suggest intentions and actions take on the role of plans, such that there is no longer a requirement for any explicit plans above and beyond intentions and actions.

Figure 1 shows the proposed relationship between explanations, intentions, and actions. Perceptions arrive from the external world, and are processed to arrive at an explanation. As shown in the diagram, this process is responsive, rather than pro-active. This is because, during this stage, the agent is making sense of what has occurred in its environment, rather than making any plans concerning what action to take in the future. However, while the explanation derivation stage is responsive, it is *not* passive. This is because the perceptions are not accepted at face value and used as the basis for the explanation. Rather, the perceptions are taken as one possible source of data concerning the agent's environment. The version of events concerning the environment which will eventually be accepted depends on results of the explanation derivation process which, far from being passive, requires varying amounts of processing, depending on the ease with which new data can be integrated into the agent's existing ontology.

Extending the design of the agent to allow the representation and manipulation of intentions renders the agent *pro-active*, as well as responsive. This is because the agent can use its intentions to represent long-term goals which it must use some degree of planning to bring about.

As shown in Figure 1, we have placed intentions at a higher level within the ontology than beliefs concerning particular actions. This reflects the fact that intentions are more centrally held than beliefs about actions. Small day-to-day occurrences cause us to constantly revise our planned actions without affecting our longer-term intentions.

For example, my intention this morning, like most mornings, was to get to work. This intention is normally executed by constructing a plan which involves me walking to work. However, this morning I looked out of the window and noticed that it was raining heavily. I therefore revised my plan, in that I decided to drive to work instead of walking, but my intention was unchanged. The reason my intention was unchanged was that this was a regular event that I knew from experience how to deal with. However, if I had experienced a highly unusual event this morning, such as an absence of gravity, it is possible that this would have caused me to revise not only my current plan but also my intention.

Our suggestion, then, is that the agent should be able to distinguish situations which do warrant the revision of intentions from those that do not. Furthermore, the agent should, on the whole, exhibit a reluctance to alter its intentions. This is because the act of intention reconsidering consumes resources which could be used elsewhere, so the agent should be discouraged from intention reconsideration except when this becomes necessary.

3.2 The Advantages of High-Level Intentions

At the sensory perception level there is a great deal of low-level information available to the agent. The dependencies, implications, and meaning of this information

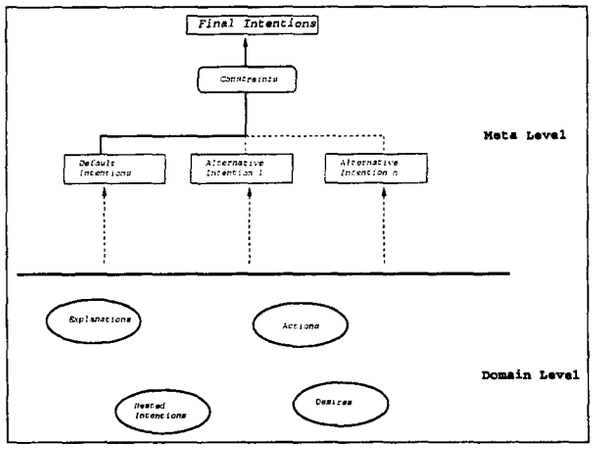


Figure 2: Intentions Provide Guidance Through the Action Search Space

are not contained in the low-level perceptual data. This makes it difficult for the agent to build a complex model solely on the basis of this information. However, high-level explanations *do* contain the connections between various pieces of sensor data which the agent must refer to when constructing its model.

Thus, high-level explanations provide methods of combining this low-level data in different ways, depending on which explanation is chosen. If the agent is able to, it will use the default explanation. If this is not possible, alternate explanations are generated and holistically compared.

The central idea that will be put forward in this section is that, within the coherence framework described in Section 2, *intentions can be viewed as the pro-active equivalent of explanations.* Competing alternative explanations are used to guide the backward chaining process which eventually yields a maximally coherent interpretation of the agent's environment. Similarly, competing alternative intentions are used to guide the planning process, and will yield a set of intentions which are maximally coherent with the agent's existing beliefs.

Just as dealing with low-level sensor data alone makes perception difficult, so dealing with low-level actions alone makes planning difficult. By considering its actions at the intention level, the agent is better equipped to deal with dependencies and inconsistencies that may exist between its planned actions.

This is not to suggest that the agent will have to generate a competing set of intentions on the basis of every interpretation of the environment. Indeed, the ability of this model to provide a well-defined yardstick by which to determine when intention reconsideration is necessary is part of its appeal.

Rather, the default interpretation will always be carried forward from the previous interpretation. If nothing has changed, the agent can go ahead and use its default set of intentions. Even if the interpretation has changed, an agent may still be able to use the default intentions. Whether or not this is possible will depend on whether or not any of the constraints which link the interpretations and intentions have been violated. If they have not, then the default set of intentions is still valid. If constraints have been violated, then the intention reconsideration process must begin.

3.3 Representing Intentions

An intention encapsulates the actions required to bring it about. It is assumed that the actions encapsulated within an intention are internally coherent. This assumption allows us to verify the coherence of the pro-active side of the agent only at the level of the agent's intentions.

Intentions are combined to form a rooted intention tree, similar to the explanation tree described in Section 2. The paths from the root intention to any of the leaves represent alternative paths through the intention space.

Intentions incorporate the following concepts:

Action Templates A data structure containing templates which describe how actions can be formed which will bring about the intention.

Preconditions States that must obtain in the world before the actions can be implemented.

Postconditions States that will obtain in the world once the actions have been implemented.

Constraints A list of constraints which are associated with the intention.

While the concepts of constraints and preconditions may seem similar, they serve different functions within the intention reconsideration process. The purpose of the preconditions and post-conditions is to guide the backward chaining process. They do this by allowing the backward chaining engine to match the preconditions of a required intention with the post-conditions of an intention which must precede it. In this manner executable intention paths can be formed.

While preconditions are used during the intention forming process, constraints are used to *verify* the consistency of an intention set with respect to the given explanation. Preconditions are used to allow the agent to combine intentions in an effective manner, and as such will usually concern lower level aspects of the agent's environment than will be represented in constraints. However, this does not mean that constraints and preconditions will necessarily be independent. Indeed, as they both concern representations of a state of affairs that must obtain before another state of affairs can obtain, there may be some similarities between preconditions and constraints which the agent will be able to exploit.

3.4 Intentions Give Meaning to Actions

Intentions combine to form coherent methods of achieving the desired goal. They are then translated into actions which are put into effect to achieve the desired intention. Note that while individual intentions are meaningful, individual actions have little meaning when taken in isolation from their related intentions. This reflects the distinction between human intentions and actions.

For example, consider my intention to type the word "for". At the intention level, I formulate the intention to type the word. At the action level, however, the actions required to carry out this intention are *all* of the form"Move hand horizontally to a certain position, then move fingers vertically."

The point is that very little meaning can be attached to actions in themselves. In order to determine the meanings of actions, we must refer to the intentions which the actions represent an attempt to bring about. This means that when reasoning about actions, we should be reasoning at the *intention* level, rather than the action level.

Grosz, Hunsberger, and Kraus argue that agents operating in a multi-agent environment can hold intentions without knowing *themselves* how to bring the intention about (Grosz et al., 1999). While this definition of intention may not be completely compatible with ours, we do agree that actions and intentions are distinct, mutually supporting entities. Thus, we are not arguing that intentions should be seen as *replacing* actions, as actions will always be necessary in order to bring about the agent's intentions. Rather, we suggest that agents should construct and manipulate plans at the intention level, rather than the action level.

Intention constraints are used by the system to determine whether variations between intended and actual execution have rendered the current set of intentions obsolete. If they have not, then intention reconsideration is not necessary. If they have, then the agent must reconsider its intentions.

Whether or not the intention constraints succeed, action constraints are applied prior to executing each action. Action constraints are designed to detect cases where the overall intention is still valid, but the original action which was originally associated with that constraint must be varied. This variation is entirely local to the current intention, and does not affect the rest of the system. In effect, this mechanism adds a degree of responsiveness to the system.

3.5 Representing Norms of Behaviour

In (Hexmoor and Beavers, 2002), Hexmoor and Beavers discuss extending the traditional BDI approach to agent design by also considering the concepts of values, obligations, and norms. They conclude that adding these new modalities increases the robustness and flexibility of the agents that are produced.

We take values to be abstract beliefs that guide the behaviour of an agent, while norms are specific instances of behaviour. In this paper we incorporate norms directly into the agent architecture, but we consider obligations and values to be implicitly represented.

The relationships between norms and intentions are represented using relations and constraints. Relations are used to represent the fact that an agent should behave in a certain way in a particular situation, and in a different way in a different situation.

It is also worth noting that norms are directly related to intentions, as opposed to actions. This is because, as described earlier, very little meaning can be attached to low-level actions, so relationships concerning the social make-up of agents must be defined at the intention level.

For example, consider an agent who takes goods from a store without paying for them. Whether or not this agent is violating any norms or values depends entirely on what its intention was. If the agent was intending to steal the goods, then it has clearly violated a norm of behaviour that would be associated with the value that stealing is wrong. If, on the other hand, the agent was intending to take the goods home on approval, and was acting with the consent of the store owner, then no norms or values have been violated.

When combined with the explanation-based backward chaining mechanism described above, this framework allows the agent to construct intention trees which are based on the adoption of a set of default norms. The constraint violation detection mechanism will ensure that any events in the environment which are inconsistent with the default model will be flagged as such, and will cause the model to be updated accordingly.

4 Implementation and Experiments

In order to test the architecture described in this paper, a system capable of constructing, executing, and manipulating intention trees was implemented in Prolog.

4.1 Experimental Domain

The experimental domain that was used was chosen to be as simple as possible while nonetheless requiring that intention trees be constructed and modified as appropriate. The domain that was chosen to be the basis of these experiments was that of a crude discgolf simulation. Discgolf is similar to golf, but players throw plastic discs instead of hitting golf balls. The environment consists of a rectangular area. At the beginning of the simulation, the disc is placed on a tee. The agent must formulate an intention tree which will permit it to throw the disc into the basket using as few throws as possible.

An advantage of this domain is that while the distinction between intentions and actions is clear at the conceptual level, at the implementational level translation from the intention level to the action level is very simple. Intentions concern an attempt to throw a disc to particular

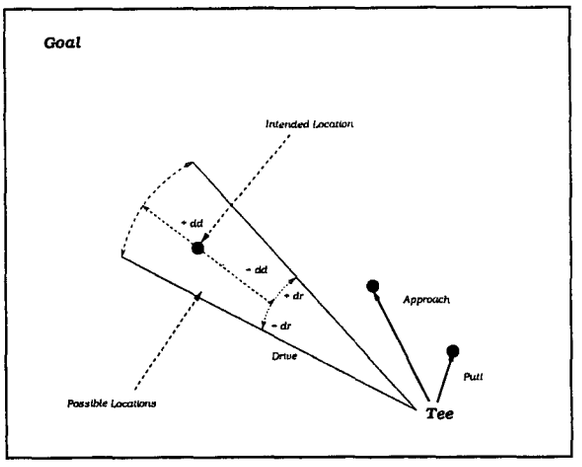


Figure 3: The Different Intentions Available to the Agent

location, specified by (x, y) coordinates. The action required to bring about the intention concerns throwing the disc in a particular direction and aiming to cover a particular distance. Deriving the distance and direction values, given the target (x, y) values and the disc's current position, is a matter of simple geometry.

The agent has three types of intention available to it, as summarised in Table 1. The intentions represent different types of throw. Throws which will move the disc further are less accurate, while the more accurate throws do not cover as much distance. This effect was achieved by adding a certain amount of error to the agent's intended throw. This is illustrated in Figure 3. The representation of the possible actual locations of the disc after a throw are based on Shanahan's concept of the circle of uncertainty (Shanahan, 1996).

The amount of error added to each throw is controlled by two values.

- *dr* represents the number of degrees by which the actual direction of the throw may vary from the intended direction.
- *dd* represents the number of units by which the distance of an actual throw may vary from its intended distance. This figure represents a percentage of the actual intended distance. For example, if the intended distance of a throw is 300 units, and $dd = 5\%$, then the actual distance of the throw will be accurate to within ± 60 units.

The *dr* and *dd* values used for the experiments described here are given in Table 1. As these experiments represent a proof of concept rather than an attempt at a realistic simulation, *dr* and *dd* were set to the same value in each experiment.

As described above, agent **CX** forms intention-level plans by constructing intention trees. Once the agent has constructed the intention tree, the agent begins to formulate actions which will bring about the intention. It does this using the action template associated with each intention. In this domain, all intentions were associated with the `throw` action. This implements the requirement that intentions should constitute meaningful segments of the agent's plan in their own right, while individual actions need not be meaningful when considered in isolation from their associated intentions.

Thus, intentions take the form:

```
[drive,[x,y]]
```

where x and y are the coordinates of the disc's intended location *after* the intention has been executed. For leaf intentions, this value will be the coordinates of the target, while for intermediary intentions, the intended disc position will be a function of the maximum distance of the intended throw and the disc's position prior to the intention.

Intention	Max Distance	*dr* and *dd* for Experiment		
		1	2	3
Putt	20	0	0.5	1
Approach	200	0	1	2
Drive	300	0	1	5

Table 1: The Intentions Available to the Agent in the Discgolf Simulation

Actions, on the other hand, have the following format:

```
[throw,D,B]
```

where D is the distance of throw required and B is the bearing at which the throw should be aimed in order to reach the position specified by the intention. Thus, the distinction in semantic level between intentions and actions is clear: intentions concern x, y coordinates in the environment, while actions concern the strength and direction of a throw.

The constraints associated with the intentions summarised in Table 1 are used to ensure that inaccuracies in throws do not necessarily lead to a change in intention. However, if the error placed on a throw is large enough, intention reconsideration may become necessary. This is achieved by calculating the distance from the disc's current position to the sub-goal specified by the intention. If this distance is greater than the maximum range of the current intention, then the current intention, and all the intentions which follow it, must be reconsidered.

The constraints associated with actions are used to allow the agent to cope with minor variations between the intended and actual course of events which do not require intention reconsideration. If the actual position of the disc prior to the current intention is different from the intended position, but still within the range of the intended throw, then a new action is created. This action will re-use the throw type specified by the intention, but will calculate distance and direction values based on the disc's actual position.

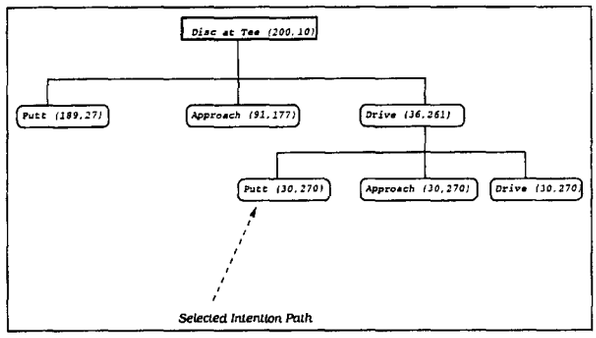


Figure 4: The Intention Tree Constructed In Experiments 1 and 2

5 Results

The description of the experiments is divided into two sections. Section 5.1 describes three single-agent experiments, while Section 5.2 describes a multi-agent experiment. The results presented in this section have been rounded to 0 decimal places in the case of (x, y) coordinates, and 1 decimal place in the case of directions.

5.1 Single Agent Experiments

Experiment 1 did not involve any errors. As such the throws associated with each intention landed with 100% accuracy, meaning there was no need for intention or action reconsideration. The intention tree constructed during Experiment 1 is shown in Figure 4.

With *dr* and *dd* set to 0, throws at the action level represented these intentions with 100% accuracy. The intentions and actions used in the experiments are shown in Table 2. Note that both throws used in Experiment 1 share the same direction, as the disc is moving along a perfectly straight line from the tee to the goal.

Exp.	Intention			Action - Throw	
	Type	x	y	Distance	Direction
1	Drive	36	261	300	56.8
	Putt	20	270	11	56.8
2	Drive	36	261	300	56.8
	Putt	20	270	10	43.7
3	Drive	36	261	300	56.8
	Approach	20	270	23	28.8

Table 2: Intentions and Corresponding Actions from Experiments 1,2, and 3

The purpose behind Experiment 2 was to investigate the ability of the architecture to formulate new actions without intention reconsideration. In order to do this a small amount of error was added to each throw, as summarised in Table 1.

Constraints associated with each action are used to ensure that the action originally represented by the intention is still valid. In this case, this was done by checking whether the disc was actually at the location it should be at when the action is undertaken.

The experiment was successful, in that the agent was able to move the disc from the tee to the goal without reconsidering its intentions, despite the fact that the disc never landed exactly where it was intended to. As would be expected, as the error added to the throws was produced randomly, results varied between different runs. The execution of the final throw, which will usually be a putt, was unsuccessful in some cases. In these cases, a new intention had to be created in order to accomplish the goal. In cases where the second throw was successful, the intention tree resulting from Experiment 2 was exactly the same as that resulting from Experiment 1, shown in Figure 4.

Results from a representative run of Experiment 2 are given in Table 2. The first intention and action are carried out as normal. After the first throw, the agent realises that the disc is not at the intended location, namely (36, 261). However, the distance between the disc's actual location and the intended location of intention 2 is such that a putt is still an applicable intention. However, as the disc is not where it should be, the original intention must be brought about using a different action.

In the example shown, the putt required by intention 2 was successful. In cases where this putt was not successful, an additional putt intention was generated, as follows:

```
[putt,[20,270]]
```

The intention is unchanged, as the goal is still that of placing the disc in the target. This intention will be translated into an action, such as:

```
[throw,2,3.4]
```

This process is repeated until the throw is successful. The actual parameters of the throw will clearly vary depending on where the disc lands after each attempt. The agent usually required between 2 and 4 throws to reach the target.

This corresponds to the approach humans take when attempting to bring about a difficult intention. For example, when trying to unlock a door at night, my intention is to place the key in the lock. My first action is to move the key toward the lock, and to attempt to insert it. If my first attempt fails, my intention *remains* that of placing the key in the lock, while at the action level I am moving the key in various directions until I feel that the key has entered the lock.

The purpose of Experiment 3 was to add sufficient error to the throws to cause the agent to reconsider its intentions. Whether or not intention reconsideration was necessary was represented using the following constraint: If the distance between the disc's actual position and its

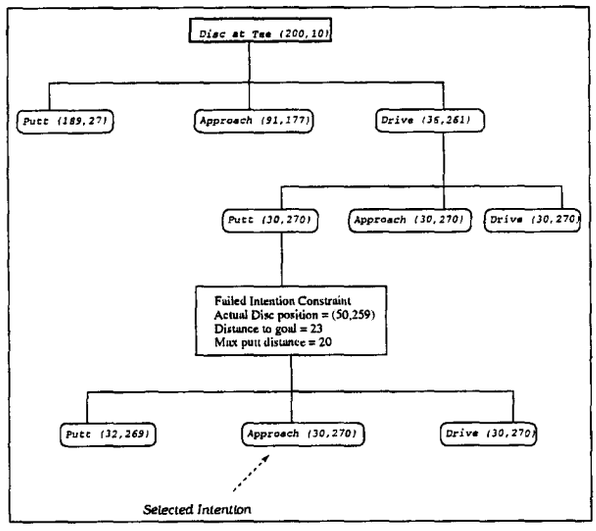


Figure 5: The Intention Tree Constructed In Experiment 3

intended position was greater than the maximum distance of the intended throw, then intention reconsideration was necessary.

An intention tree representing a sample run of Experiment 3 is shown in Figure 5. For the sake of simplicity, a run has been selected here in which the agent successfully completed the task in 2 throws. Most of the runs in Experiment 3 required between 3 and 5 throws.

After the first throw, the disc ends up at $(50, 259)$. This is too far for the intended putt, so intentions must be reconsidered. Using the disc's actual position as the starting point, the agent produces a new set of intentions. A putt is indeed too short to reach the target, but an approach throw may be successful. The approach intention is selected, and translated into a distance and direction value, as shown in Table 2.

Despite the relative simplicity of the domain, these experiments show that it is possible for an agent to construct an intention tree in order to bring about a long-term goal. A small amount of variation when executing an intention will not necessarily require intention reconsideration, but the agent will be willing and able to reconsider its intentions on the fly if this becomes unavoidable.

5.2 Extending the Experiments to the Multi-Agent Scenario

Agent **CX** is able to model and represent the intentions of other agents. The explanation-based component of the agent uses observations of the actions of other agents to form the most coherent explanation of what the other agents in the environment are attempting to do. Once the intentions of the other agents have been decided, agent **CX** can adapt its intentions accordingly. The precise nature of this adaptation will depend on the application domain, and on the relationship between the agents.

In cooperative discgolf, two players play on the same team. Both players make a throw, and then decide which was the best throw. Both players then play from the overall best throw on the previous shot, and so on. If the first player plays a good safe shot, then the second player is free to attempt the risky shot, as the team has nothing to loose. On the other hand, if the first player's shot is unsuccessful, the second player must also attempt a safe shot.

In our simulation, agents playing cooperatively may play in accordance with two norms of behaviour, namely `safe` and `risky`. In the multi-agent scenario, each of the three intentions described in Section 5.1 is associated with either a safe or a risky norm of behaviour. Safe intentions have lower *dr* and *dd* values and a shorter range, while risky intentions have higher *dr* and *dd* values and a higher maximum range. The *dr*, *dd*, and maximum range values associated with the two norms in Experiment 4 are shown in Table 3.

Intention	**Max Distance**		***dr* and *dd***	
	Safe	Risky	Safe	Risky
Putt	20	40	1	5
Approach	200	300	2	10
Drive	300	400	5	20

Table 3: The Intentions Available to the Agents in Experiment 4

As in the single agent case, both agents construct intention trees. As shown in Table 3, the likely range and error values for each intention depend on whether the agent has adopted a `safe` or `risky` norm of behaviour for that shot. Both agents construct their intention trees assuming that they will be using the `safe` norm. If conditions arise which allow an agent to play under the `risky` norm, this condition will be detected by a constraint violation, and the agent will revise its intention tree appropriately.

A feature of this framework is that as long as all agents in an environment share the same ontology, they will all construct *exactly* the same intention trees.

This is due to the fact that this framework is not based on probabilities. The operation of the intention tree construction and revision mechanism may be complex, but it is completely deterministic and repeatable. This means that agents sharing the same starting ontology, and receiving the same perceptions from the environment, will construct the same intention trees. This allows multi-agent cooperation in complex domains without the need for inter-agent communication.

The cycle used by the two agents is as follows:

1. Determine which of the two players, **A** or **B**, made the best throw on their last turn. This is determined by measuring the distance form both discs to the target. Store the best disc location in (bx, by)

2. Both **A** and **B** now play from (bx, by). The agent

that made the best throw on the previous turn plays first.

3. The agent to play first makes their throw. The first agent to throw will always be using the `safe` intentions.

4. The second agent to throw observes where the first agent's disc landed. Based on this information, it has two choices:

   - If the first throw was good, the team of agents now has nothing to lose. This means that the second agent to throw is free to adopt a risky style of play, and so alters its intention tree so as to select the appropriate `risky` intention.
   - If the first throw was not good, the second agent must play a safe shot, rather than a risky one.

Turn	Agent	Intention			
		Type	Norm	x	y
1	**A**	Drive	Safe	36	261
	B	Drive	Risky	30	270
2	**A**	Approach	Safe	30	270
	B	Approach	Risky	30	270
3	**A**	Putt	Safe	30	270
	B	-	-	-	-

Table 4: Intentions and Norms From Experiment 4

Results from a representative run of Experiment 4 are given in Table 4. The explanation for these results is as follows: At the start of play, agent **A** threw first, using a `safe` norm of behaviour. The disc ended up at position $(63, 254)$. This qualifies as a good throw, meaning that agent **B** is now free to adopt a `risky` norm for the same throw. This is reflected by the fact that agent **B** is intending to throw the disc all the way into the basket on the first throw.

Agent **B**'s first throw is less successful, ending up at $(157, 50)$. The distance from agent **B**'s throw to the target is 254, compared with a distance of 36 for agent **A**'s throw, so both agents move to $(63, 254)$.

Agent **A** throws first on turn 2, as it threw the best shot on the previous turn. The distance to the target is 36, which is within the range of a safe approach or a risky putt. However, as this is the first shot for this turn, agent **A** has no choice but to play the safe approach. Agent **A**'s shot ends up at $(27, 273)$, which is only 4 away from the target. This also qualifies as a good shot, so agent **B** is once again free to play a risky shot. Agent **B**'s shot ends up at $(43, 317)$, which is considerably worse than agent **A**'s shot, so both agents move to $(27, 273)$. Agent **A** now faces an easy putt, and hits the target on the first attempt.

These results show that agent **B** was able to change its style of play, depending on the results obtained by its team-mate. In this instance, the performance of the team was not helped by agent **B**'s risky shots, but it is nonetheless clear to see why agent **B**'s choice of actions was correct, as the potential gains for the team outweighed the potential losses for the individual.

5.3 Experimental Results - Summary

Our interest in this domain stems from the fact that while it is relatively simple to implement, it nonetheless yields the following interesting features:

- Agents can operate either individually or as part of a team. The best course of action for an individual agent is not necessarily the best course of action when the agent is operating as part of a team. Agent **CX** is able to modify its behaviour when operating as part of a team, as an agent playing individually would never be wise to adopt a risky style of play.
- The agents are homogeneous. Each agent can intend to play in one of two different way. Which norm the agent chooses to adopt depends on the actions of its team-mate. As shown above, agent **CX** is able to dynamically adapt to new norms of behaviour at run-time.
- The agents can plan pro-actively at the intention level, can react to small variations in disc position at the action level, and are able to adapt to different styles of play depending on the actions of their team-mate.

6 Acknowledgements

This work is supported by AFOSR grant F49620-00-1-0302.

7 Conclusions

In this paper we have shown how an agent architecture which was based on philosophical approaches to truth and justification can be extended to produce a framework for intelligent agent design. We have shown that agents based on this framework are capable of responding to events in the environment and of exhibiting goal-directed pro-active behaviour.

In addition, this framework facilitates the design of agents which are capable of forming robust intention-level plans which can cope with small variations in the environment without necessitating plan regeneration. When the environment changes to such an extent that a new plan is necessary, the agent is capable of detecting this situation and constructing a plan which addresses the problem

it was facing. A feature of this architecture is that multiple agents can cooperate on the same problem without communicating, as they will each form identical intention trees as long as they share the same starting ontology.

Thus, we conclude that the proposed framework can be used to design and implement situated adaptive agents which can exhibit responsive, pro-active, and co-operative behaviour in a multi-agent environment.

References

Robert Audi. *The Structure of Justification.* Cambridge University Press, Cambridge, UK, 1993.

Barbara Grosz, Luke Hunsberger, and Sarit Kraus. Planning and acting together. *AI Magazine*, 20(4):23–34, 1999.

Henry Hexmoor and Gordon Beavers. In search of simple and responsible agents. In *Proceedings of the GSFC Workshop on Radical Agents*, MD, 2002.

Nicholas R. Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *International Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

Richard L. Kirkham. *Theories of Truth: A Critical Introduction.* The MIT Press, Cambridge, Mass, 1995.

Nicholas Lacey. *Investigating the Relevance and Application of Epistemological and Metaphysical Theories to Agent Knowledge Bases.* PhD thesis, University of Wales, Aberystwyth, 2000.

Nicholas Lacey and Mark Lee. The implications of philosophical foundations for knowledge representation and learning in agents. In Daniel Kudenko and Eduardo Alonso, editors, *Proceedings of the AISB 01 Symposium on Adaptive Agents and Multi-Agent Systems*, pages 13–24, York, UK, 2001. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour.

John Pollock. Planning Agents. In Rao and Wooldridge, editors, *Foundations of Rational Agency.* Kluwer, 1998.

Willard Van Orman Quine. Two dogmas of empiricism. In *From a Logical Point of View : 9 logico-philosophical essays*, chapter 2, pages 20–46. Harvard University Press, Cambridge, Mass, 1980.

Martijn Schut and Michael Wooldridge. Principles of intention reconsideration. In *AGENTS'01*, Montreal, Canada, 2001. ACM.

Murray Shanahan. Noise and the common sense informatic situation for a mobile robot. In *AAAI/IAAI, Vol. 2*, pages 1098–1103, 1996.

Paul Thagard and Elijah Millgram. Inference to the best plan: A coherence theory of decision. In A. Rom and D.B. Leake, editors, *Goal-driven learning*, pages 439–454. MIT Press, Cambridge, MA, 1997.

Peter van Inwagen. *Metaphysics.* Dimensions of Philosophy Series. Oxford University Press, Oxford, 1993.

Michael Wooldridge. *Reasoning About Rational Agents.* MIT Press, Cambridge, MA, 2000.

Regular behaviour of learning agents in Minority Games

Alexei Lisitsa; Igor Potapov
Department of Computer Science,
University of Liverpool
Liverpool L69 7ZF, U.K.
e-mail: {A.Lisitsa, I.Potapov}@csc.liv.ac.uk

Abstract

We consider deterministic and non-deterministic variants of the well-known minority game (D.Challet and Zhang (1997); Challet et al. (1999)), in which k (where k is odd) agents repeatedly choose between 0 and 1, attempting to make a *less popular* choice at every step. Agents make their decisions based on finite depth history of the game and equipped with a simple learning mechanism which chooses the "most successful so far" strategy from the fixed pool of strategies. In this paper we restrict ourselves to the case of a *non-virtual evaluation* principle, that is, after each turn only the strategy which *actually* has been applied is evaluated by an agent. We show that in both variants the game as a whole generates rather restricted classes of behaviours: *ultimately periodic* in the deterministic case and *regular* in the non-deterministic one.

1 Introduction

The minority game was introduced by D.Challet and Zhang (1997) as a variant of the earlier El-Farol problem (Arthur (1994)). It is a multi-agent competitive decision game model which despite its simplicity demonstrates non-trivial behaviour and has been used for modelling the dynamics of financial markets[1], see (D.Challet and Zhang (1997); Challet et al. (1999)) and references therein.

The standard variant of the minority game can be formulated as follows.

At every time step $0, 1, \ldots$, (of a linear discrete flow of time) k agents (with k being odd) choose between two possibilities, say, 0 and 1. An agent wins a round if his/her choice is among less popular at this time step. The standard variant of the game assumes that the only information on the result of any particular round made public is the most popular choice. Standard variant assumes also that agents make decisions based on the recent record of results in the following way. Every agent has l possible finite m-depth strategies, each of which gives a definite decision once provided with last m results and, at any given time the agent plays the strategy that has been most successful up until that point of time. The ability to use the best strategy provides a simple learning mechanism.

The above description of the learning mechanism left several points unspecified.

Firstly, different strategy evaluation principles are possible. The classical one, which we refer to as *virtual evaluation*, is "after each turn, the agents assign one (virtual) point to each of his strategies, which would have predicated the correct outcome" (D.Challet and Zhang (1997)). So in this case *all* the strategies are evaluated on every turn. Alternatively, one can have a *non-virtual evaluation* principle, that is, after each turn only the strategy which *actually* has been applied is evaluated by an agent. In this case the strategy can be not only rewarded one point, but also be penalized one point.

Secondly, there are different approaches on how to choose a strategy if there are several equally successful strategies at some moment ? A standard proposal was (D.Challet and Zhang (1997); Challet et al. (1999)) to use *random* choice. Since a majority of research papers on minority games dealt with *statistical properties* of games, this proposal was quite natural. From a computational point of view a *deterministic* and *non-deterministic* policies are reasonable alternatives.

In the present paper we study *behavioural* properties of minority games from a *computational* viewpoint and consider the games with

- non-virtual strategy evaluation principle, and
- deterministic and non-deterministic policies of choice between equally successful strategies.

Surprisingly, it turns out that minority games under these assumptions demonstrate structurally simple behaviour. When we started this work we were expecting computationally more complex behaviour, since the learning mechanism is implemented by a set of counters, and it is well-known that even two counters with simple control is enough to obtain Turing complete devices (Minsky (1967)). But restrictions on control imposed by minority games

[1] Intuitively, the relevance of minority games to market's dynamics can be understood from simple observation: both to sellers and buyers usually *it is good to be in minority* – if there are a few sellers the prices are high, which is good for sellers, while being one of the few buyers is good as well for keeping prices low.

and its specific learning mechanism in fact lead to regular behaviour.

2 Preliminaries

The minority game can be seen as a particular kind of collective competitive game between learning agents interacting via an environment. At any round of a game every agent receives a binary word as input from an environment and responds with binary decision (0 or 1).

The environment, having collected the responses of all agents, produces for every agent: 1) the information on what was the winning decision (0 or 1), and 2) a word as the new input.

Every agent, in turn, having received its result, updates (if necessary) its strategy or chooses new strategy and applies it for new input, producing new output, ... and so on.

More specifically we have the following

Definition 1 *A learning agent α with l-strategies of depth m or simply, an (l,m)-agent is defined by the set of l strategies $s_1,\ldots,s_l$ of depth m, which are finite functions: $s_i : \{0,1\}^m \to \{0,1\}$, $i = 1\ldots l$.*

With every strategy s_i an agent has an associated integer-valued counter, storing $c_i(t)$, the value, or rating, of this strategy at the round t.

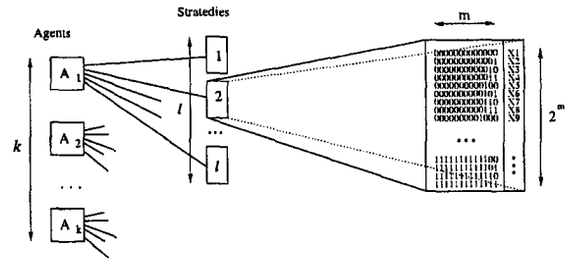


Figure 1: System of (l,m)-agents

We assume that every (l,m)-agent has a strategy selection function $f : Z^l \to \{1,\ldots,l\}$, $f(c_1,\ldots,c_l) = \{j | c_j = max(c_1,\ldots c_l)\}$, which given the current rating of the strategies yields the subset of (indices of) the most successful strategies so far.

A *non-deterministic* agent chooses a strategy from the set of the most successful strategies non-deterministically.

In the *deterministic* case we assume an agent has a preference relation (linear order) on its strategies, and it is convenient to assume that all its strategies are listed according to this order, i.e. s_i is more preferable than s_j iff $i < j$. Having that, a deterministic agent always chooses a strategy with minimal index among the most successful ones.

At any round an agent applies its selected strategy to the input received from the environment and outputs the result.

The learning mechanism of every agent is implemented by a *rating update* rule:

- If an agent has applied the strategy s_i at the round t and won then $c_i(t+1) = c_i(t) + 1$
- If an agent has applied the strategy s_i at the round t and lost then $c_i(t+1) = c_i(t) - 1$

Before we define the notion of minority game we need to introduce the notion of an *initial history* (pre-history) [2] I of the length $m > 0$ as a sequence $I(1),\ldots,I(m)$ of $0's$ and $1's$ of length m. It is convenient to have an alternative notation of $H_{[-m+1,0]} = H_{-m+1},\ldots,H_0$ for a pre-history.

Definition 2 *A standard (non-)deterministic minority (k,l,m)-game γ is given by an initial history I of the game of a length m, k (non-)deterministic learning (l,m)-agents, where k is odd, and an environment E, which executes four consecutive actions in each time step t:*

1. send a word w that is a history $H_{[t-m,t-1]}$ to all (l,m)-agents;

2. receive an output $x_1,\ldots,x_k$ from all agents;

3. compute a minority *function of odd arity k $minor_k$: $\{0,1\}^k \to \{0,1\}$ defined as follows:*

- $minor_k(x_1,\ldots,x_k) = 1$

$$if\ |\{x_j | x_j = 1\}| < |\{x_j | x_j = 0\}|$$

- $minor_k(x_1,\ldots,x_k) = 0$

$$if\ |\{x_j | x_j = 0\}| < |\{x_j | x_j = 1\}|$$

and communicate the result as the winning choice to all agents.

4. update the history,

$$H[0,t] = H[0,t-1] \cdot minor_k(x_1,\ldots,x_k)$$

A (k,l,m)-game γ generates an *evolution* of the game, which consists of two parts: history H_γ of the game and personal attendance[3] histories A^t_γ of each agent α_t $(1 \le t \le k)$.

H_γ is an infinite sequence of 0 and 1, $H(i) = 0(or 1)$ means that 0 (or 1) is a winning decision at the moment i. Denote by $H_\gamma[i,j]$ an interval of history between moments i and j:

$$H_\gamma[i,j] = [H(i),\ldots H(j)].$$

An attendance history A^t of the agent α^t is an infinite sequence of 0 and 1. $A^t(i) = 0$ (or 1) means that the t-th agent plays 0 (or 1) at the moment i.

[2] Since the actions of agents depend on m-depth history of the game we need some convention of how to start game; it seems the simplest decision is to make the initial m-history just a part of the definition of the game.

[3] This terminology came from the early El-Farol problem setting, where decision 1 is interpreted as "to go to the bar", and 0 – "not to go".

3 Periodicity of deterministic minority (k, l, m)-game

Consider a local state of an agent in a deterministic minority (k, l, m)-game at some round t. Let $c_1(t), \ldots, c_l(t)$ be the values of its counters, describing the current strategies ratings. Short analysis of the principles of non-virtual evaluation and deterministic choice shows that we have the following property of the counter values: there exists $j, 1 \leq j \leq l$ such that

$$\forall p\ (p < j \Rightarrow c_p(t) < c_j(t))$$

$$\forall p\ (p > j \Rightarrow c_p(t) \leq c_j(t))$$

$$\forall p, q\ (p < j \wedge q < j \Rightarrow c_p(t) = c_q(t)) \qquad (*)$$

$$\forall p, q\ (p > j \wedge q > j \Rightarrow c_p(t) = c_q(t))$$

$$\forall p, q\ (p > j \wedge q < j) \Rightarrow c_p(t) = c_q(t) + 1);$$

see the figure 2 a).

Moreover, the future agent behaviour depends actually only on the current strategy at the given round and on its *relative value*. The relative value of a current strategy s_j at the round t is the difference between $c_j(t)$ and $c_j(t')$, where $t' < t$ is the round, from which the agent started to apply s_j continuously. This suggests to describe the behaviour of the agent by an one-counter automaton, where the counter contains a relative value of a current strategy and the states correspond to the different strategies, see the figure 2 b).

Thus we will use the pair $\langle j, c(t)\rangle$ to represent a *local state* of an agent. Here j is the number of the current strategy and $c(t)$ is the relative value of the strategy.

The *global state* of the (k, l, m)-game at the round t is a k-tuple of the local states of the agents

$\langle j_1, c^1(t)\rangle, \ldots \langle j_k, c^k(t)\rangle$ together with a current m-depth history $H_{[t-m,t-1]}$.

Given a (k, l, m)-minority game, one of the participating agents is said to be *unbounded* iff for at least one of its strategy s_i the rating of this strategy is unbounded during the play, i.e. $\forall M \exists t |c_i(t)| > M$

Any infinite sequence $a_1, a_2, \ldots$ is said to be *ultimately periodic* if it is periodic starting from some point, i.e. $\exists i > 0 \exists j > 0\ \forall t.(t > i) \rightarrow a_{t+j} = a_t$

The main theorem of this section is as follows

Theorem 1 *Any standard deterministic minority (k, l, m)-game has ultimately periodic behaviour, that is each agent generates ultimately periodic attendance history, and the game has ultimately periodic history.*

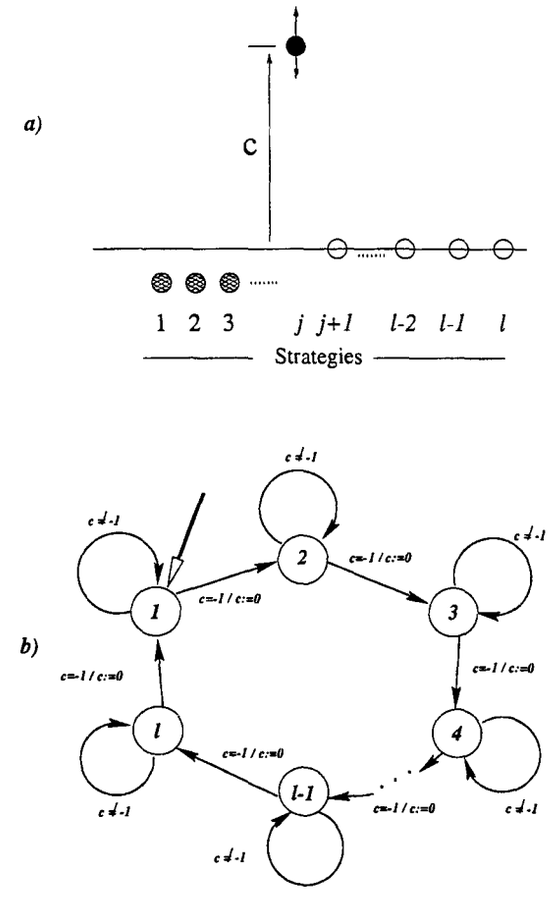


Figure 2: a) possible values of counters in a deterministic agent b) the transition graph of a deterministic agent

Proof.

Case 1. There are no unbounded agents. Then there are only finitely many global states in the game. It follows that some global state repeats and the game has ultimately periodic behaviour.

Case 2. There is precisely one unbounded agent A. Then there exists an infinite sequence $I_1 = [a_1, b_2], I_2 = [a_2, b_2], \ldots I_r = [a_r, b_r], \ldots$ of the intervals on the time flow with the following properties:

$\forall i\ (c(b_i) - c(a_i) \geq i)$

$\forall i, j\ (j > i \Rightarrow c(a_j) > c(a_i))$

$\forall i, x\ (x \in (a_i, b_i) \Rightarrow c(a_i) < c(x) < c(b_i)$

We call such a sequence the sequence of *fair* intervals for A, see figure 3.

As the all other agents are bounded there are only finitely many local states they can went through. Furthermore, the number of m-depth histories is also finite and bounded by 2^m. Taking M big enough, we will have on the interval I_M that the entire game repeats its global state, except, possibly, the relative value of the unbounded strategy s_j of the agent A. By further increasing the M one can guarantee that the value of unbounded strategy increases between the repeating points on I_M. It follows

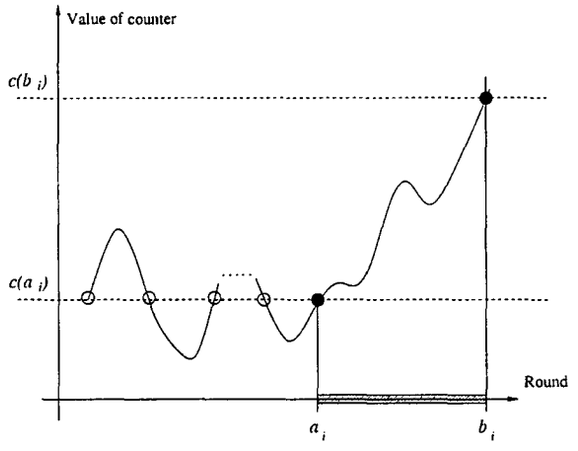


Figure 3: Fair interval for one agent

that A will always apply the same (unbounded) strategy s_j and the entire game is ultimately periodic.

*Case 3.*There are two unbounded agents A and B. Construct the sequence of fair intervals $I_1, I_2, \ldots, I_r \ldots$ for A. The behaviour of the agent B on this sequence falls into one of the following subcases:

a.) The agent B is uniformly bounded on the sequence, that is the rates of all its strategies are bounded by some constant K.

b.) The agent B is unbounded on the sequence, and there is a such subsequence $I_{i_1}, I_{i_2}, \ldots, I_{i_r}, \ldots$ and the sequence $J_1, J_2, \ldots J_r, \ldots$ of fair intervals for B such that $\forall j; J_j \subseteq I_{i_j}$.

c.) The agent B is unbounded on the sequence, but the condition of the case b.) is not satisfied.

Subcase a.) is treated similarly to the *Case 2*, as on the sequence of fair intervals for A we have one unbounded agent, that is A.

Subcase b.) is straightforward generalization of the *Case 2*. The sequence $J_1, J_2, \ldots J_r, \ldots$ is fair for both A and B. Taking M big enough one can enforce to repeat the global state on I_M, except, possibly, the values of two unbounded strategies of A and B . Futher, by choosing appropriate M one can guarantee that between the repeating points the values of the strategies increase. It follows that A and B will keep applying the same strategies forever and the entire game is ultimately periodic.

Subcase c. In that subcase one can choose a subsequence $g = a_{j_1}, a_{j_2}, \ldots, a_{j_r}, \ldots$ of the starting points of the intervals $I_1, \ldots I_r, \ldots$ such that for the values of unbounded strategies c^A and c^B of the agents A and B, respectively, we have $i' > i \Rightarrow (c^A(i') > c^A(i) \wedge c^B(i') > c^B(i))$. Consider the global states of the game on this subsequence only. Take a long enough segment of g, the global state of the game repeats, except, possibly, the values of c^A and c^B, but these values are monotonically increasing on g. It follows that A and B will keep applying the same strategies forever and the entire game is ultimately periodic.

Case 4. There are three, or more unbounded agents. This is done by straightforward, but tedious extension of above arguments and use of induction on the number of unbounded agents. We show that

- either, at least one of the agents is bounded on the sequence of fair intervals for A, then the argument similar to that in the Case 3 a) reduces the problem to that with less number of unbounded agents; or
- all unbounded agents have a common system of fair intervals and the arguments similar to those in the Case 3 b) are applied; or
- one can choose a long enough sequence on which the values of all unbounded strategies monotonically increase and then the argument similar to that of the Case 3 c) is applied.

Detailed description will appear in the full paper. □

4 Non-deterministic minority games.

A history tree H_γ of non-deterministic (k, l, m)-game γ is an infinite labelled tree $H_\gamma = (V, A, r, C)$, where the set of vertices V is the set of all global states of γ, A is a set of arcs :

$$\{\langle x, y\rangle | x, y \in V \text{and transition from } x \text{ to } y \text{ is possible}\},$$

r is an initial global state of γ, $C : V \to \{0, 1\}$ is a colouring of vertices: $C(v) = 1$ if 1's won at the state v and $C(v) = 0$ otherwise.

A history tree H_γ is finitely branching. The vertices with branching degree > 1 could appear due to non-deterministic choice of (possibly several) agents. It is easy to see that a branching degree of H_γ of any (k, l, m)-game γ is bounded by l^k.

A vertex v of H_γ is said to be a *branching point* iff its branching degree > 1. Two branching points v_1 and v_2 of H_γ are said to be *neighboring* iff all intermediate vertices on the (unique) path from v_1 to v_2 are not branching points (i.e. have a branching degree $= 1$).

A *branching type* $\tau(v)$ of any vertex v is a tuple

$$\langle S_1, \ldots, S_k\rangle,$$

where k is a number of agents and $S_i \subseteq \{1, \ldots, l\}$ is a subset (of indices) of applicable strategies for i-th agent in the state v, i.e. those strategies with maximal rates. Notice that if v is not branching point, then all S_i in $\tau(v)$ are singleton sets.

The tree H_γ induces a partial order $\leq$ on its vertices: $v_1 \leq v_2$ iff there is a path from v_1 to v_2 in a tree.

We define a binary relation $\rhd$ on subsets of $L = \{1, \ldots, l\}$:

$$S \rhd S' \Leftrightarrow$$

$$\exists x \in L(x \notin S' \wedge S = S' \cup \{x\} \wedge S' \neq \emptyset) \vee$$

$$\exists y \in L(S = \{y\} \wedge S' = L)$$

The relation $\rhd$ can be seen as a kind of reduction, working on subsets of L : one can reduce the set S to S' by removing one element if there is more than one element in S; and if there is only one element in S one can "jump" to the entire set $S' = L$

We use $\rhd$ in the following key lemma which reveals the dynamic properties of non-determinism in the the presence of the minority principle.

Lemma 1 *For history tree H_γ of any non-deterministic (k, l, m)-game γ and any two neighbouring branching points $v \leq v'$ in H_γ with branching types*

$$\tau(v) = \langle S_1, \ldots, S_k \rangle \text{ and } \tau(v') = \langle S'_1, \ldots, S'_k \rangle$$

the following holds:

$$\vee_{i=1}^{k} (S_i \rhd S'_i)$$

Proof.(hint) The main idea is to use the properties of the counters values, analogous to those $(*)$ of the deterministic agents.

□

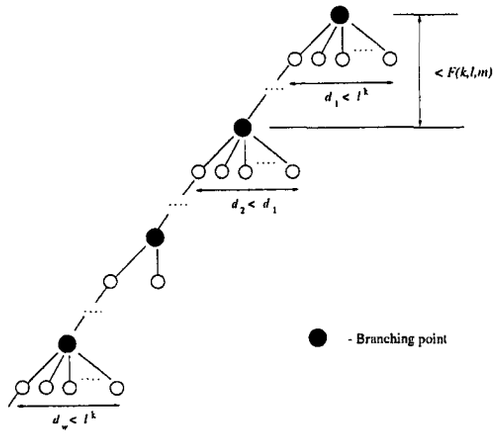


Figure 4: History tree for a non-deterministic Minority Game

The next lemma shows another crucial property of history trees

Lemma 2 *In a history tree H_γ of any (k, l, m)-game γ the distances between any two neighbouring branching points are bounded by a finite constant, depending only on k, l, and m.*

The figure 4 illustrates the statements of Lemma 1 and 2.

The main theorem of this section is as follows

Theorem 2 *The history tree H_γ of any non-deterministic minority (k, l, m)-game is regular, i.e. there exists only finitely many non-isomorphic subtrees in H_γ*

The proof of the theorem is based on Lemmas 1 and 2 and on the property of m-depth dependency of labelling.

5 Conclusion and future directions.

We have shown that the combination of a simple learning mechanism, which evaluates strategies and applies the best strategy (deterministically or not), with the strategies of finite depth produces rather restricted class of regular behaviours in minority games.

We conjecture, however, that even in the presence of random choice some kind of regularity [4] of possible histories holds.

A regularity of the behaviour means that many natural problems on minority games are decidable. For example, periodicity of history for deterministic games implies existence of a limit of the ratio *number of wins/number of rounds* with number of rounds going to $+\infty$ for any agent, and the problem "who is a winner, in the limit, if any ?" becomes decidable. The questions of decidability will be considered elsewhere.

We intend to extend our investigation of the behaviour of variations and generalizations of minority game with:

- virtual evaluation principle;
- more powerful strategies;
- more general learning policies; and
- different winning conditions.

As to the more powerful strategies, our initial computational experiments suggest that modest extension by adding some counting abilities [5] to the agents probably does not lead to non-periodic behaviour.

And finally, a look at learning agents within the framework of minority games from the viewpoint of Formal Learning Theory (Jain et al. (1999)), especially along the lines of (Montagna and Osherson (1999)), seems very promising for understanding their learning abilities.

Acknowledgements

We would like to thank Michael Fisher, Grant Malcolm and anonymous referees for valuable comments that helped us improve the presentation. A. Lisitsa was supported by EPSRC grant GR/M46631.

[4] Perhaps, formulated in terms of probabilistic finite automata.

[5] For example, checking "what choice was most successfull so far" – this can not be implemented by bounded depth strategies.

References

W.B. Arthur. Inductive reasoning and bounded rationality. *Economic Assoc Papers and Proc.*, 84:406–411, 1994.

D. Challet, M. Marsill, and R. Zecchina. Theory of minority games. 1999. Preprint, http://xxx.lanl.gov/cond-mat/99014392.

D.Challet and Y.-C. Zhang. Emergence of cooperation and organization in an evolutionary game. *Physica A*, 246:407, 1997.

Sanjay Jain, Daniel Osherso, James Royer, and Arun Sharma. *Systems that Learn, Second Edition*. M.I.T. Press, Cambridge MA, 1999.

M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall International, 1967.

Franco Montagna and Daniel Osherson. Learning to coordinate: A recursion theoretic perspective. *Synthese*, 118(3):363–382, 1999.

"To do or not to do": The Individual's Model for Emergent Task Allocation

Kurt Schelfthout; Tom Holvoet
K.U.Leuven, Departement Computerwetenschappen, Celestijnenlaan 200A, B-3001 Leuven, Belgium;
Kurt.Schelfthout@cs.kuleuven.ac.be; Tom.Holvoet@cs.kuleuven.ac.be

Abstract

Task allocation is one of the most important aspects in multi-agent systems. *Dynamic* task allocation makes multi-agent systems more adaptable and hence robust to change in the environment. Several approaches for dynamic task allocation have been proposed. Most are based on models of social insect behaviour. In this paper, we propose a new model for dynamic task allocation. This model integrates and generalizes concepts of existing approaches, and aims to offer more flexible and adaptable agents. Experiments with a proof of concept application yield promising results.

1 Introduction

Multi-agent systems face above all the challenge of task allocation. An adequate assignment of tasks to agents is essential for achieving the overall behavior of the system. Research in multi-agent design and negotiation recognize the importance of this fact and as such, they aim to solve the problem statically. An ideal allocation is either left up to the programmer or user by defining roles for agents (Brazier et al., 1997; Zambonelli et al., 2001), or it is based on ideal or heuristic solutions to an optimization problem which results in predefined interaction protocols (Zlotkin and Rosenschein, 1989; Kraus, 2001).

These static approaches lack flexibility and adaptability that are essential to cope with dynamic aspects of the environment. A dramatic change in the environment or in the constitution of the multi-agent system (in some systems, agents may disappear or new agents can join) leads to unpredictable or erroneous results, or leads to extra complexity in the negotiation protocols.

Dynamic task allocation approaches allow agents to change the tasks they are performing during execution. Based upon their perception of the state of the multi-agent system (global goal vs. global solution so far), their observation of the environment and other agents, agents may decide to adopt another task. In these approaches, the task allocation problem is an individual agent's problem (to choose which of the tasks it should perform on the one hand, and to determine how and when a task switch may occur on the other hand).

Our research focuses on reactive multi-agent systems with emergent overall behavior (Bonabeau et al., 1999; White and Pagurek, 1999; Bonabeau et al., 1998a; Parunak, 1997). Reactive agents typically have a particular set of tasks (often called "behaviors") that they can perform. These tasks are triggered by stimuli from the environment (either through stigmergy or through direct interaction with other agents), or from within the agent (internal stimuli). Stimuli are also the triggers for agents to adapt their behavior, and as such allow for adaptivity in the multi-agent system.

In this paper, we present a new model for the emergent task allocation of an agent. Existing models are e.g. the threshold model and the social inhibition model. The model we propose allows more adaptability in that it allows a fine-grained selection, although still based on stimuli only. It generalizes from previous models, allows the integration of different existing models and thus allows more feedback from the environment and better use of available information. As such, it overcomes the problems known in existing models for task allocation.

The task allocation model is inspired by the biological world of insects, and is the result of a detailed literature study of social insect behavior (Bonabeau et al., 1999, 1998b; Dornhaus et al., 1998; Detrain et al., 1999; Tofts, 1991; Parunak, 1997; Beshers and Fewell, 2001; Deneubourg and Goss, 1989). We evaluated the model on a concrete MAS application, a distributed mail retrieval problem, where agents have to decide when and where to retrieve mail from customers located in different zones (Bonabeau et al., 1998b). The new model proved to be more controllable and adaptable to user needs than the original threshold model.

The rest of this paper is organized as follows. In section 2 we review some existing models. In section 3 we explain why a new model is needed, and propose one. Section 4 presents some results we obtained using the new model. Finally, we draw some conclusions, and indicate some directions for future work.

2 Emergent task allocation

To achieve emergent task allocation, the basic idea is to associate a stimulus with each task (Bonabeau et al. (1999, 1998b)). This stimulus is an indication of how important it is for the agent to execute the task. Agents can add stimuli of their own, to recruit other agents for the task. Stimulus level is thus an indication of the urgency of a task. For example, hunger can be a stimulus to go searching for food (a task). We will use some more examples from biology, and more specifically, from ants, throughout the text.

A number of concrete architectures that use these ideas, have been proposed. Most ideas emerged in the field of biology, where researchers increasingly try to explain social behaviors of animals based on simple, local rules in an individual. We briefly review three important contributions.

A first important model is the *threshold model* (Bonabeau et al., 1999, 1998b; Dornhaus et al., 1998). Here every agent has a number of response thresholds, that indicate its sensitivity to a number of stimuli, that are in turn connected to a task. The lower an agent's response threshold, the more likely it is to respond to a given stimulus. These response thresholds are fixed, but can be different for individual agents. It is then possible to define a number of agents that respond preferentially to one kind of stimulus. E.g. an ant with a low response threshold for 'hunger' becomes a forager, another tends to the brood.

A variant of the *threshold model* incorporates learning, and accounts for task specialization. In this model, whenever an agent executes a task, the response threshold for that task is decreased. Whenever it is not executing the task, the threshold is increased. Therefore, agents that often execute a certain task are more sensitive to the stimuli associated with that task. This model is more robust to perturbations of the system than the fixed threshold model, because generally agents respond to tasks faster, and so stimulus level decreases faster.

A second model is the *social inhibition* model (Huang and Robinson, 1999). Here an activator increases the chance that an agent will execute a certain task (associated with the activator), while likewise an inhibitor decreases this chance. As an example, define an internal activator for foraging that increases with age. Define an inhibitor that is passed from forager to another ant whenever they interact. Now, when an ant does not interact with any foragers, it starts to forage at a certain age. However, when it interacts a lot with other foragers, it is inhibited (possibly because there are enough agents already on the job) and will do something else for a longer period of time.

A third model is the *foraging for work* (FFW) model (Tofts, 1991). This model incorporates a spatial arrangement of tasks. Each task is put along a production line, where the input for one task is the output from another. Agents continue to do any task for which there is a need. When there is no longer a need to execute the task in a given zone, agents seek a new task in nearby zones. This fairly simple model has been controversial among biologists, but generates interesting results: some agents appear to be specialists, continually working in the same zone, while others appear to be more generalist, filling up the empty spots in various zones. When mortality is introduced, and agents are "born" at one end of the production line, it can be shown that they drive the older workers to more outward zones, so that a form of temporal specialization emerges: older agents tend to do tasks in outward zones, while younger agents stay closer to the "nest".

We have reviewed these three models because they each emphasize one particular aspect of MAS that we find important. Firstly, there is the aspect of what *the agents' internals* look like. These affect how the agent decides, how it incorporates experience etcetera. The threshold model is a powerful and elegant way to model this. A second aspect is the *social behavior* aspect: how agents interact with other agents. This is modeled partly by the social inhibition model. Finally, there is the agents' *environment*, emphasized by the FFW model in the spatial distribution of tasks. More models have been developed, that together provide a more complete picture. We refer to Beshers and Fewell (2001) for a complete overview as well as critiques of these models.

3 A task selection model

All these models were conceived to explain animal behavior, and as such attain the goal of simulating a behavior that is close to an aspect of the observed real animal behavior. However, we argue that most of the ideas in these models can be generalized and combined to provide more adaptive forms of task allocation that are useful in MAS. Furthermore, generalizing some of these ideas allows a better understanding of the actual concepts underlying these models, and allows a better understanding of the interplay between different parameters. Last, a more general model that can implement all of the aforementioned models makes it easier for us to compare and rate alternatives. We now describe our proposal for a first step in this direction (see Figure 1).

We choose thresholds as the central concept, because this is a powerful, simple yet elegant representation of an agent's willingness to execute a certain task. We choose the concept of stimuli as the means to influence the agents' immediate (short-term) decisions, as well as a means of communication between agents and their environment. The concept of activators and inhibitors is used to allow agents or environment to influence agents' decisions in the long term.

Looking at figure 1, stimuli are the means by which an agent views the world. Of course, it doesn't see all stimuli in its environment all the time - this is limited by its view on the world. It then processes these perceptions internally using thresholds, activators, etc. The outcome

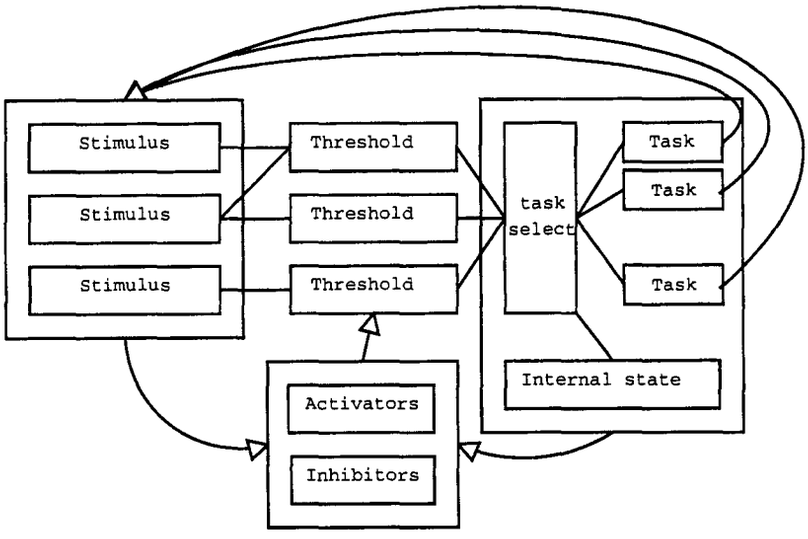


Figure 1: A conceptual task selection model. Important concepts are in boxes, arrows indicate influences.

of this internal process is a certain task, which can be seen as the agent's actuators, it's means of changing the world. The most obvious effect of executing a task should be a decrease in the appropriate stimulus. We will now discuss these various concepts in more detail.

Stimuli are the only way for an agent to get signals from the world. There are three kinds of *external* stimuli: (1) environmental stimuli the agent nor any other agent has put there (e.g. a food source), (2) direct stimulus by interaction with another agent, (3) stigmergic stimuli, communication from other agents through the environment (e.g. a scent trail). The difference between these three is not visible for the agent itself. Internal stimuli are internal drives that make the agent do a certain task (e.g. hunger makes it search for a food source).

Thresholds are the central part of the task selection mechanism. Each threshold has at least one stimulus. Multiple stimuli must be combined into one, using some function of the stimuli values (e.g. maximum, sum,...). Each threshold also has one task associated with it. A low threshold indicates that the agent is likely to execute the task associated with that threshold. A threshold activates its corresponding task whenever its stimulus, or a combination of stimuli, has exceeded the threshold's value.

The *task selector* chooses one of the activated tasks (each task has a threshold associated with it), and executes it. It chooses based on the relative values of the stimuli that lead to activation of a task, and based on the *internal state* of the agent.

Internal state is necessary to check which tasks are useful to execute in the current situation. For example, it is possible that the execution of a certain task has left the agent in a state so that it cannot execute some other task (e.g. if it is already holding some food, it cannot pick up any more).

Activators and *inhibitors* modify the threshold values. The idea is to generalize and combine the varying threshold model and the social inhibition model. An activator/inhibitor can be triggered in two ways: by stimuli, and by the execution of a certain task. For example, specialization can be achieved by lowering the response threshold (using an activator) for the task that is currently being executed.

As can be seen from the model, there are two possible feedback mechanisms: one to influence the thresholds (through the activators and inhibitors), and another through the execution of the tasks themselves. This combination allows a faster response to stimuli, as well as better control, and more mechanisms to influence other agent's behavior. The MAS thus proves to be more adaptive, both to short term and long term changes.

4 Experiments and Results

This model was implemented as a Java framework. The generic framework can be instantiated to a task selection component in an agent. As a proof of concept, we experimented with agents in a simple environment, that of distributed mail retrieval (Bonabeau et al., 1998b). In this problem, agents are allocated to pick up mail in different zones of a city, while overall demand for mail pickup should be kept as low as possible (see figure 2).

Translating this problem to our model, stimuli for agents are the demands from the various zones. The agents' tasks are: go to a particular zone, and pick up the mail in that zone. Doing so reduces the demand in the zone to zero. Our test bed consist of a grid of 5x5 zones, with five mail agents randomly located on the grid. Every step, the demand in five randomly chosen zones is increased. Every agent can then decide what its next move will be. The order in which the agents decide on every step is random.

During experiments, we studied two aspects of the problem. First, the *average demand* over all steps. Obviously, this should be as low as possible. Second, we looked at how the agents divide work: which zones does

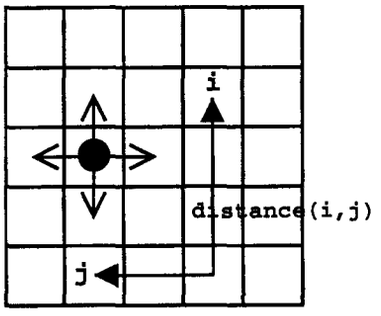


Figure 2: A grid of 5x5 zones. Agents cannot move diagonally. Distance is Manhattan distance.

Parameter	description	Value
α	demand coefficient	500
θ	initial threshold	500
θ_{min}	minimal threshold	1000
θ_{max}	maximal threshold	0
ϵ_0	learning coefficient	-150
ϵ_1	learning coeff neighbors	-70
ϕ	forgetting coefficient	50

Table 1: Parameters for the experiment

each individual agent visit, and how many *conflicts* occur - a conflict occurs when one agent is decreasing demand or traveling to a zone another agent is also traveling to or decreasing demand in. Over a certain number of experiments, the percentage of conflicts was calculated as follows. For each run, all the agent's moves were logged. On each step, we count one conflict if two agents move to the same destination. The percentage of conflicts is then calculated as $\frac{n_{conflicts}}{n_{agents} \cdot n_{steps}}$.

4.1 Experiments with a Self-Reinforcement Model

As reference experiments, we ran some tests with a "standard" reinforcing threshold model (Bonabeau et al., 1998b). That is, each agent has 25 tasks (one for each zone). The stimulus $s(i,j)$ it receives from a certain zone (i,j) is calculated as follows:

$$s(i,j) = \frac{\alpha * demand(i,j)}{demand(i,j) + distance(i,j)}$$

where $demand(i,j)$ is the demand in the zone with coordinates (i,j), and $distance(i,j)$ is the distance from the agent's current position to that zone. Whenever an agent picks up mail in a zone, an activator decreases the threshold for that zone with value ϵ_0, and its neighboring zones with value ϵ_1. Agents will thus become specialists for a certain area of the map. Thresholds for zones not tended to increase slowly with value ϕ (agents will "forget" they have been there). For a description of these and other parameters, see table 1.

We did various experiments in this setting, and checked both conflicts and average demand in runs of 1000 experiments, consisting of 10000 steps each. For the parameters detailed above, we found a mean demand of 1986 with a 95% confidence interval [1924, 2048]. For the percentage of conflicts we found a mean of 0.0606, with a 95% confidence interval $[0.059611, 0.061658]$. This means that about 6% of the time agents are going to a zone that is already looked after. We would obviously like to reduce these numbers significantly.

4.2 The Influence of the Agents' Stubbornness

In the above experiments, agents could not change their mind about going to a particular zone while actually traveling to it. So basically, when there is a demand in one zone only, most agents will decide to move to it. When the first agent reaches the zone and resets the demand to zero, all others still traveling will not even observe the fact that there is no longer a demand in that zone. They will only reassess the situation when their task is done. To attempt to alleviate this problem, we introduced a parameter p - the probability at each step that an agent changes his mind, and reassesses the situation. So, the lower p, the more stubborn the agent becomes. For $p = 0$, the agents would be the same as in section 4.1.

We ran tests with $p = 0.25$. So on average every four steps an agent will look at the world and reassess the situation. We hoped this would allow the agents to respond to fluctuating demand faster. However, results prove the contrary: for $p = 0.25$ we found a mean demand of 1914, 95% confidence interval [1847.09, 1981.89], which is not a significant difference. Tests with $p = 1$, meaning that an agent can reassess where to go on every step, also had no significant influence on the average demand. Both these results were found over 1000 runs consisting of 10000 steps each.

This observation can be explained as follows. Although the less stubborn agents observe the world more often, and base their decision on more recent information, this also means that they make more detours to reach a target zone. This is because they can change their minds underway to a certain zone, and thus sometimes have to go back. This offsets their ability to react to changes faster: they become somewhat indecisive.

4.3 Marking Territory

In a further attempt to improve performance, we added extra stimuli for the agents to respond to. Every time an agent decides to move to a certain zone, it will drop a stimulus indicating that it is going to that zone. These stimuli have a certain strength that decreases with time. When an agent encounters such a stimulus in a zone where it happens to pass, the threshold for the task that stimulus was dropped for will increase. In other words, when

an agent encounters a trail to a certain zone, it will be less likely to go to that zone in the future. The idea is that agents will mark their territory, and by doing so drive other agents away from it. This will result in less conflicts, and consequently less average demand.

Again, we ran 1000 experiments of 10000 steps each. We found a mean conflict percentage of 0.0411, where the real mean is in [0.0405, 0.0417] with 95% confidence. Comparing this to the results from section 4.1, it is obvious that this is a statistically significant decrease. More information is transferred between the agents, and an extra negative feedback loop is entered into the system, what leads to better decisions. The performance of the system as a whole also increased: mean demand is 1589, with a 95% confidence interval [1548, 1629].

5 Conclusion and Future Work

The contributions of this paper are as follows. First, we defined task allocation problems as a task selection problem for each individual agent. We advocated that using a form of emergent task allocation is more adaptive than other forms. We then identified a few classes of possible concrete solutions to emergent task allocation, and suggested an architecture for an agent that combines and generalizes from these models. Finally, we presented some promising results with a proof of concept application.

We believe that this research will contribute to a thorough understanding of forms of emergent task allocation, which is still lacking in current state of the art. This will bring the application of emergent task allocation a step closer. Obviously, more tests are needed on more challenging problems, to evaluate and push to the limits of the model's possibilities. Furthermore, a more formal analysis of its properties needs to be done, to increase our understanding of the interplay of various parameters.

References

Samuel N. Beshers and Jennifer H. Fewell. Models of division of labor in social insects. *Annual Review Entomology*, 46:413–440, 2001.

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)*, 1998a.

E. Bonabeau, A. Sobkowski, G. Theraulaz, and J. Deneubourg. Adaptive task allocation inspired by a model of division of labour in social insects, 1998b.

Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. SFI Studies in the Sciences of Complexity. Oxford University Press, 1999.

F. M. T. Brazier, B. M. Dunin-Keplicz, N. R. Jennings, and J. Treur. DESIRE: Modelling multi-agent systems in a compositional formal framework. *Int Journal of Cooperative Information Systems*, 6(1):67–94, 1997.

J.-L. Deneubourg and S. Goss. Collective patterns and decision-making. *Ecology, Ethology and Evolution*, 1: 295–311, 1989.

C. Detrain, J.L. Deneubourg, and J.M.Pasteels, editors. *Information Processing in Social Insects*. Birkhäuser, 1999.

A. Dornhaus, F. Klügl, F. Puppe, and J. Tautz. Task selection in honey bees - experiments using multi-agent simulation. In *Proc of GWAL'98*. Verlag Harry Deutsch AG, 1998.

J. Ferber and A. Drougoul. Using reactive multiagent systems in simuation and problem-solving. In L. Gasser and N. Avouris, editors, *Distributed Artificial Intelligence: Theory and Practice*. Kluwer Academic Publishers, 1992.

Zhi-Yong Huang and Gene E. Robinson. Social control of division of labor in honey bee colonies. In Detrain et al. (1999), pages 165–182.

Sarit Kraus. Automated negotiation and decision making in multiagent environments. In *Multi-agent systems and applications*, pages 150–172, 2001.

H. Van Dyke Parunak. "Go to the Ant": Engineering principles from natural multi-agent systems. *Annals of Operations Research*, 75:69–101, 1997. Special Issue on Artificial Intelligence and Managment Science.

Chris Tofts. Task allocation in monomorphic ant species. Technical Report ECS-LFCS-91-144, Department of Computer Science, University of Edinburgh, 1991.

T. White and B. Pagurek. Emergent behaviour and mobile agents. In *Proceedings of the workshop on Mobile Agents in the Context of Competition and Cooperation at Autonomous Agents '99*, 1999.

F. Zambonelli, N. R. Jennings, and M. Wooldridge. Organizational abstractions for the analysis and design of multi-agent systems. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*, volume 1957 of *Lecture Notes in AI*. Springer-Verlag, January 2001.

Gilad Zlotkin and Jeffrey S. Rosenschein. Negotiation and task sharing among autonomous agents in cooperative domains. In N. S. Sridharan, editor, *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 912–917, San Mateo, CA, 1989. Morgan Kaufmann.

Generative Migration of Agents

F.M.T. Brazier; B.J. Overeinder; M. van Steen; N.J.E. Wijngaards

Department of Computer Science, Faculty of Sciences, Vrije Universiteit Amsterdam;
de Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
{frances,bjo,steen,niek}@cs.vu.nl

Abstract

Agents, and in particular mobile agents, offer a means for application developers to build distributed applications. In current agent systems, mobility of agents is constrained by the environment of the agents: the agent platform (which supports agents) and the agent's code base (e.g., DESIRE, Java). Generative migration is needed to adapt an agent to conform to its destination agent platform and code base. In this paper generative migration is described as a process of "transparently adapting" an agent. An agent can continue to function at its new location on a completely different agent platform.

1 Introduction

Agents, and in particular mobile agents, offer a means for application developers to build distributed applications. In current agent systems, mobility of agents is constrained by the environment of the agents: the agent platform (which supports agents) and the agent's code base (e.g., DESIRE, Java). Within the same agent platform and code base, agent migration has been shown to be possible. However, many agent platforms exist, differing substantially in the support for agents. Write once - run everywhere is not yet true for agents...

This heterogeneity of agent platforms, combined with heterogeneity in code-bases of agents, leads to an interesting question concerning agent mobility: can an agent migrate across heterogeneous platforms? The answer is relatively simple, as an agent needs to be adapted to fit its destination agent platform and code-base.

In this paper generative migration (see Brazier et al. (2002)) is described as a process of "transparently adapting" an agent. Section 2 presents an overview of the principles behind agent factories: the entities responsible for processing blueprints. Section 3 describes the use of the agent factory for generative migration. Section 4 investigates implications of generative migration for agents and agent factories. Section 5 discusses transparent adaption by generative migration.

2 Agent Factory

An agent factory is a facility that creates, and modifies, software agents, see Brazier and Wijngaards (2001). It can be used to adapt agents so that they can use specific programming languages and run on different agent platforms. The design of an agent within an agent factory is based a *blueprint*. The blueprint of an agent contains a configuration of conceptual building blocks which specifies the agent's functionality and behaviour. The blueprint also contains one or more configurations of detailed building blocks, which specify an operationalisation of the conceptual functionality and behaviour.

A mapping is defined between building blocks at conceptual level and building blocks at detailed level. (Note that this mapping may be structure preserving, but that this is not necessarily ideal.) A detailed description of a building block includes the operational code. For each conceptual description, a number of detailed descriptions may be devised and vice versa. These detailed descriptions may differ in the operational language (e.g., C++, Java), but also in, for example, the efficiency of the operational code. The conceptual descriptions may differ in the modelling paradigm (e.g., UML, DESIRE), but also in, e.g., the detail in which an agent's functionality is modelled.

Building blocks themselves are configurable, but cannot be combined indiscriminately. The open slot concept is used to regulate the ways in which components are combined. An open slot in a component has associated properties at both levels of abstraction that prescribe the properties of the building block to be "inserted".

A prototype of the agent factory automatically (re-)designs an information retrieval agent: its blueprint and executable code. The blueprint of an existing simple information retrieval agent is briefly described by Brazier et al. (2002). This prototype agent factory itself is written in Java, and contains enough knowledge to (re-)design simple information retrieval agents.

3 Generative Migration

One of the strengths of the agent factory concept is that it provides a means to support migration of agents in heterogeneous environments. Section 3.1 discusses pre-conditions for successful migration of agents. Section 3.2 describes the approach in agent-factory-enhanced migration. Section 3.3 describes migration scenarios.

3.1 Migration pre-conditions

A mobile agent is simple an agent having the ability to move between different machines, e.g. see Tanenbaum and van Steen (2002). Agent migration entails transfer of both the agent's executable code and state. The concept of generative migration is geared to weak mobility: parts of the state of the agent are migrated to another host. Strong mobility, i.e. migrating running processes (including their memory usage, stack, heap, etc.), is not possible with generative migration, as in most cases agent executables change.

Generative migration requires (1) agent factories, (2) implementation independent representations of agent functionality, and (3) implementation independent formats of agent's state (e.g., XML, RDF or OIL may be used, see Horrocks et al. (2001)).

Assuming that both the source and the destination host both have access to an agent factory (for simplicity's sake), these agent factories need to have building blocks with comparable functionality. One solution is that the agent factories share the same libraries of conceptual building blocks, but each have different libraries of detailed building blocks. Another solution is to have conceptual building blocks in ZEUS (see Nwana et al. (1999)) and DESIRE (see Brazier et al. (1997)) with comparable functionality.

3.2 Migration using the agent factory

Migration using an agent factory diverges from standard mobility of agents in that it is *not* executable code with state that is migrated, but instead the agent's blueprint together with (parts of) the agent's state. Consider the following scenario for heterogeneous mobility (also described by Brazier et al. (2002)), depicted in figure 1.

An information retrieval agent A currently resides on host machine $H1$. This host runs the Ajanta agent platform, developed by Tripathi et al. (1999), and supports Java agents. The agent wants to move to another host: host $H2$. Host $H2$ runs the DESIRE platform, and its agents run code generated by the DESIRE execution environment, see Brazier et al. (1997).

In the process of migrating the agent A from host $H1$ to host $H2$, the agent first needs to store information on its (mental) state. Then the agent factory on host $H1$ sends the blueprint of the agent, together with the state information of the agent to the agent factory at host $H2$.

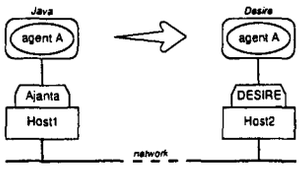


Figure 1: Example migration scenario in which agent A on Host 1 (written in Java, running on Ajanta) migrates to Host 2 (where it will be specified in DESIRE and running on DESIRE).

Host $H2$'s local agent factory receives the blueprint of the agent and state information. This agent factory constructs a DESIRE agent A on the basis of the blueprint of agent A. This DESIRE agent A (i.e., a functionally equivalent incarnation of the Java agent A) runs on DESIRE's virtual machine (the DESIRE-interpreter), and is able to incorporate information on its state.

3.3 Migration scenarios

Migration including re-generation of agents is a more complex process, requiring more resources, than migration without agent re-generation. Four migration scenarios are distinguished in Brazier et al. (2002):

- *Homogeneneous migration.* An agent migrates to another host without any changes in either the virtual machine or the agent platform. This situation is most common in practice, and does not warrant generative migration.
- *Cross-platform migration.* An agent migrates to another host with the same virtual machine, but a different agent platform. Generative migration may be used to adapt the agent to the target agent platform, e.g. by using wrapper interfaces. If both agent platforms have the same standard interface (e.g., advocated by OMG, see OMG (2000), and FIPA, see FIPA (2001)), the agent need not be adapted.
- *Agent-regeneration migration.* An agent migrates to a host with a different virtual machine, but the same agent platform. Generative migration is needed to regenerate the agent's executable code for the target virtual machine and the agent platform.
- *Heterogeneous migration.* An agent migrates to a host with a different virtual machine and a different agent platform (see the scenario described in Figure 1. In this situation, generative migration is needed to regenerate an agent for the target virtual machine and the target agent platform.

The authors are unaware of agent platforms supporting agent-regeneration migration and/or heterogeneous migration.

4 (Re)Generation versus Adaptation

Generative migration has implications for both agents and agent factories. Not only do agents need to understand the concept of *locality*, but also the concept of *incarnation*. Section 4.1 discusses implications for the role of agent factories. Section 4.2 briefly describes aspects of agent incarnations.

4.1 Role of Agent Factories

Agent factories are responsible for (re)generating an agent, while adhering to preferences of the agent and the (destination) agent platform. An incarnation of an agent needs to be designed which may be executed by a virtual machine at the target host, and which can interface with the agent platform at the target host. The agent (re)generative process is responsible for minizing the quantity and quality of the changes to the agent. The process of regenerating an agent mainly depends on available libraries of building blocks.

The agent regeneration process is facilitated by the two levels of abstractions distinguished within the agent factory: conceptual and detailed. In general, agent factories need to have building blocks with comparable functionality. In the general situation, a configuration of conceptual building blocks needs to be constructed. Agent factories sharing common libraries of conceptual building blocks (with equal functionality) is a specific case, providing common ground to the agent factories involved. In this case, the configuration of conceptual building blocks need not be changed. If an applicable configuration of detailed building blocks is present for the target host, the agent factory only needs to assemble this configuration into operational code. In this case the adaptation is *transparent* to the agent: the agent need not be aware of the fact that it has another incarnation than before.

When no suitable configuration of detailed building blocks is available, a configuration of detailed building blocks may need to be (re-)designed. The agent factory needs strategic knowledge to decide on a configuration of detailed building blocks which minimizes the loss of functionality and services for the agent. In addition, the agent factory may provide an agent with functionality to cope with the loss of specific functionality.

An agent may be fitted with functionality for introspection, awareness of its abilities, and understanding functionality and services needed to achieve (its) goals. Such an agent can adapt its behaviour, and pursual of goals, with respect to its current incarnation.

4.2 Agent incarnations

The incarnation of an agent is entails activation of both the "code and data" of the agent in another environment (the virtual machine possibly with another interface to an agent platform.). Figure 2 depicts the concepts related to

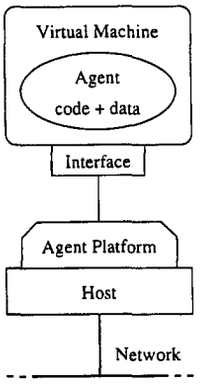


Figure 2: An agent's incarnation involves not only its code and data, but also its immediate physical environment.

the incarnation of an agent. An agent's code and data is executed in the context of a virtual machine; examples are the Java Virtual Machine and Prolog interpreters. The agent has access to an interface to its agent platform via its virtual machine. Through this interface the agent can access services provided by the agent platform, e.g. (group) communication, generative migration, etc.

The agent needs to conform to both its target virtual machine and its target agent platform interfaces. For reasons of privacy and security, it is assumed that the state of an agent does not need to be adapted: an agent can easily employ a programming language independent representation format for its state (e.g., on the basis of XML, RDF or OIL, see Horrocks et al. (2001)). The (re)generation process has the goal to generate an operationalisation of the (conceptual) functionality of the agent.

In the ideal adaptation process, the agent does not "notice" any changes to its incarnation. It still has access to all of its functionality, can resume execution from its state, and can access services provided by its current agent platform. When agent platforms, and virtual machines, differ extensively, it is up to the agent factory to mask these differences.

In less ideal situations, an agent may be aware of classes of services and functionality which may be unavailable during/for specific incarnations. The agent needs to adapt to this situation, e.g. by employing strategies for circumventing missing functionality and services (e.g., enlisting support by other agents).

An agent may specify preferences concerning its incarnation. With these preferences, an agent can specify, e.g., which functionality and/or services are of more importance to its (correct) functioning than other functionality and/or services. An agent may, in addition, specify that specific functionality and/or access to specific services may be (temporarily) unnecessary at a specific host.

The preferences specified by an agent may also state how the agent is to be informed of success or failure of generative migration. Does the agent expect a message

in a specific format? Does the agent expect information concerning its current (dis)abilities as a facts-base?

Security and trust are of importance in generative migration. The agent trusts an agent factory to generate the "right" incarnation. An agent cannot be easily protected against a malicious agent factory, which e.g. may introduce code to spy on the agent. The administrator of an agent platform trusts its agent factory and libraries to generate agents which cannot damage other agents, the agent platform or the hosts running the agent platform and agents.

5 Discussion

Mobile agents are currently restrained in their mobility by their environment. Current agent platforms expect a homogeneous environment, i.e. hosts running the same agent platform and the same virtual machine. This paper proposes an approach which transcends homogeneity of agent platforms and virtual machines: *generative mobility*. In generative mobility, a blueprint of an agent's functionality is transported, together with information on the agent's state. At its destination, an agent factory regenerates the executable code of the agent on the basis of its blueprint: a new incarnation of the agent. Upon activation, the agent may restore its state and resume execution.

Ideally, an agent factory is able to (re)generate an agent such that it retains all of its functionality and access to services: transparent adaption. However, this may not be possible in situations requiring heterogeneous migration. An agent needs to be aware of characteristics of its current incarnation, including limitations in functionality provided by its current incarnation and services offered by the current agent platform.

Generative migration is one of the services researched within the AgentScape project on worldwide scalable distributed agent operating systems. Currently a prototype of the agent factory (namely the libraries of components) is being built that supports generative mobility.

Acknowledgements

This research is supported by NLnet Foundation, http://www.nlnet.nl. The authors wish to acknowledge the contributions made by Hidde Boonstra, David Mobach, Oscar Scholten and Sander van Splunter.

References

F. M. T. Brazier, B. D. Dunin-Keplicz, N. R. Jennings, and J. Treur. Desire: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, 6:67–94, 1997.

F. M. T. Brazier, B. J. Overeinder, M. van Steen, and N. J. E. Wijngaards. Agent factory: Generative migration of mobile agents in heterogeneous environments. In *Proceedings of the AIMS Workshop at SAC 2002*, 2002. to appear.

F. M. T. Brazier and N. J. E. Wijngaards. Automated servicing of agents. *AISB journal*, 1(1):5–20, 2001.

FIPA. FIPA agent platform, 2001. http://www.fipa.org.

I. Horrocks, F. van Harmelen, P. Patel-Schneider, T. Berners-Lee, D. Brickley, D. Connoly, M. Dean, S. Decker, D. Fensel, P. Hayes, J. Heflin, J. Hendler, O. Lassila, D. McGuinness, and L.A. Stein. DAML+OIL. http://www.daml.org/2001/03/daml+oil-index.html, March 2001.

H. Nwana, D. Ndumu, L. Lyndon, and J. Collis. ZEUS: A toolkit and approach for building distributed multi-agent systems. In *Proceedings of the Third International Conference on Autonomous Agents (Autonomous Agents'99)*, pages 360–361, 1999.

OMG. Mobile agent facility specification. OMG Document formal/00-01-02, Object Management Group, Framingham, MA, January 2000.

A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, Upper Saddle River, New Jersey 07458, 2002.

A. Tripathi, N. Karnik, M. Vora, T. Ahmed, and R. Singh. Mobile agent programming in Ajanta. In *Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS'99)*, pages 190–197, Austin, TX, May 1999.

Architectural Principles for Social Influence among Benevolent Agents

Henry Hexmoor
Computer Science & Computer Engineering Department
University of Arkansas, Fayetteville, AR 72701
hexmoor@uark.edu

Abstract

Benevolent agents with a high level of autonomy can be designed to be aware of their social influences in order to make sure the overall missions are successful. Additionally, the system of social influences can be engineered to meet the designer's objectives of predictable agent behavior. In this paper we outline a few agent architecture principles for such agents.

1 Introduction

Often the objective in designing multiagent systems of *benevolent agents*[1] is a system where interaction among agents is most congruent and beneficial to the overall mission. This leaves us with a choice in design between (a) detailed protocols and policies for interaction that leaves agents with little autonomy, and (b) agents with high autonomy that are driven by their social relations to determine their own rates and types of interaction relative to their shared mission. We choose the latter, which requires agents to continually monitor and to adjust their relationships for the overall mission success. An important paradigm in agent-based systems is to consider intentional notions of Belief, Desire, and Intention (BDI agents) (Wooldridge 2000). BDI agents possess update and revision functions for each intentional component. Beliefs are adapted to the agent's current state of mind and changes in the environment. Desires or specific goals are adapted to the agent's beliefs and attuned with the changing environment. Intentions over specific actions are adapted to agent's beliefs and desires. These adaptations create adaptivity for the agent at a behavior level called action selection. Coordination with other agents provides yet another form of adaptivity. Consider a scenario where two agents A1 and A2 are building a structure of blocks as shown in Figure 1. Both A1 and A2 have access to piles of blocks and know about the structural stability of blocks. An agent can only move one block at a time. In order to preserve stability, both agents must simultaneously place blocks 3 and 4. We can specify this problem with temporal constructs of parallel actions and ordering of agent actions or by allowing one agent to dynamically use its social relationship to take a lead role and cue the other. Specifically when it comes to block positions 3 and 4, whoever is the lead agent will set the rate of placement and the follower agent will monitor the lead agent for coordination. A1 and A2 opportunistically become lead depending on which one gets the block first and they are both equally willing and likely to play lead or follower.

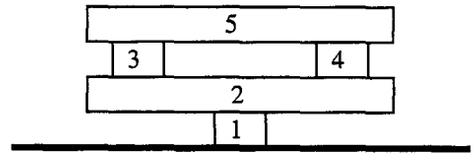


Figure 1 Blocks world

In this paper, we describe architectural principles that account for these dynamic adjustments due to social influences in the context of collaborative work. We discuss intuitively plausible types of responsibility and how agents can maintain it. In the remainder of this paper we will discuss social influences, agent relationships, and how agents can maintain the notion of the mission success. The paper ends with concluding remarks.

2 Social Influence and Relationships

As shown in Figure 2, social actions promote social influences. Since social factors are inter-related, initial social influences due to social actions might produce secondary and indirect influences. Actions of agents are guided by social values and norms, which affect the relationships among social influences. Therefore, social actions of one agent will influence the other agent in the context of prevailing norms and values as well as by the strength of the social action.

Whereas *physical actions* predominantly produce physical change, and *speech actions* predominantly produce epistemic change, *social actions* predominantly produce influence. Social action might be an action by one agent toward another, a mutual action of multiple agents, a

[1] Agents who are not benevolent have a more complex relationship to social influence and responsibility and this is beyond our current scope. Benevolent agents are characterized by their propensity to offer help.

bilateral action by multiple agents, or a group action. For brevity and illustration we limit our attention to the actions that commonly cause influences over the following notions: Help, Permission, Delegation, Service, and Resources. Actions about Help are performed to aid others in their task and specific actions might be to provide, to withhold, to request, or to reject help. Common actions over Permission are to request, to grant, to deny, and to withdraw. Actions over delegation might be to issue, to accept, or to reject. Service is an act but unlike help that is pro-active, it is passive. Common actions over service are to request, to give, or to deny. Common action over resources might be to request, to offer, to reject, to take away, or to prevent.

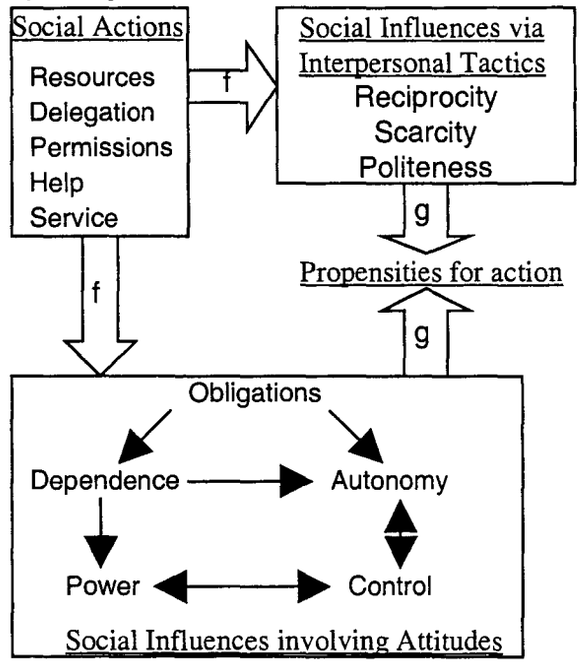


Figure 2 Social actions and influences; values and norms are not shown in the figure

The kind of influence we have in mind is close to normative (utilitarian) influence in social psychology in that agents reason about gains and losses from interpersonal interactions (Kassin 2001, Brehm, et al, 2002). Influences might involve overt interpersonal tactics such as reciprocity, scarcity, and politeness. Overt influences are immediate and deliberate as in interactions described in FaintPop (Ohguro, et al 2001). Reciprocity is when one agent returns an act by another or in effect *pays* for an act. An agent might reason about the reciprocity norm and perform a social act (e.g., help) based on an expected propensity in repayment. Scarcity is the norm that short supply produces a demand. An agent might use that norm and deny or hide services or resources. This is commonly used in theories of persuasion (Larson, 1998). Politeness as an interpersonal tactic is to get another agent to yield to another agent.

Influences might also be indirect and of indefinite duration. One type of indirect influence is via changes of attitudes. This is shown as the box in the lower part of Figure 2. These are perceived changes in social relationships that affect an agent's ties. The Figure shows our focus on Autonomy, Dependence, Obligations, Control, and Power and salient relationships we see among them. Later in this section we will discuss interdependencies among attitudes. Let's refer to the set of influences as I. Let's refer to the set of social actions as A. We define function f that maps the agent's current beliefs B, a set of currently active values V, a set N of currently active norms, and a set of social actions A to a set of influences I:

$$f: B \times V \times N \times A \rightarrow I.$$

An example is delegation of homework by a teacher to a student. Following shared norms governing relationships among teachers and students, assigning homework produce the influence for the student to adopt the obligation to carry out the homework.

Agents use influences that result from social actions they experience in their action selection. In addition to social influences, action selection accounts for means end analysis and rationality principles that are governed by the agent's endogenous sources. How action selection is affected by social influences is a complex issue that is beyond our current scope and is denoted as function g in Figure 2. Agents can project such a propensity for action in deciding to perform a social action. The reasoning might also include a chain effect where one agent produces an influence in another, which in turn produce an effect in another and so on. An agent can intend such a proliferation of influences and intentionally start such a chain reaction. This in fact is commonplace in a team setting.

We appeal to the reader's commonsensical beliefs, norms, and values to highlight a few relations between social actions and resulting social influences.[2]

1. When there is social action of request for help, the requesting agent is willing to lower its autonomy over a goal it cannot accomplish alone and would like to share or delegate the goal to someone else. An agent who finds itself in a position to respond to help is reciprocally willing to share or be delegated the goal that the requester cannot accomplish.
2. When someone has desirable resources that is somewhat scarce and announces availability of resource as a social action, there is a potential for dependence if

[2] With more time and space we will state these more formally and concisely. We are seeking conditions under which these relationships hold and our enumeration here is only to point out the salient relationships. Also, not all relationships in the Figure 2 are explained.

other agents wish to access the resource. If some arrangement is made for provision of resources in exchange for something, the agents who want the resource and are willing to exchange, in addition to dependence, may experience a lowered autonomy over their goals that requires the resource.

3. An agent, who has a service that is commonly desired and is in short supply, has a high autonomy with respect to how it will make its services available. Reciprocally, an agent who need s scarce service will experience a low level of autonomy and high level of dependence toward agent who can provide that service.
4. An agent who delegates a task to another agent and when both agents are consenting to this delegation, the delegating agent will experience a dependence about the task and the delegee will experience an obligation to carry out the task toward the delegator.
5. When an agent increases or decreases permissions it gives to another agent, it is proportionately increasing or decreasing autonomy in other agents.

Next we highlight a few interdependencies among social influences that involve attitudes. Once an initial social action produces an influence on an attitude, indirect influences are propagated via relationships among attitudes. We feel engineering agents with specific connection among social attitudes is a powerful method for tuning agents indirectly.

1. When an agent has power over another agent it has indirect control over the agent. Similarly, when an agent has control over another agent, it has indirect power over the agent.
2. Autonomy and control are complimentary. I.e., Control and autonomy add up to a total amount and one lacks in some it has excess in another.
3. Obligations induce dependence.
4. Dependence diminishes autonomy.
5. Dependence induces power.

Next, we suggest that quantities of influence are in part based on utilities of norms and values. Additionally, magnitudes of social influences experienced depend on utilities derived from intensity of the social action. M(I) will denote magnitude of an influence I. In general, there might be differences between intended influence by the agent performing the social action and the receiving agent but for simplicity we assume these are the same amounts.

3 Behavior Guarantees

The values and norms can be designed to encode the mission of the collaborative effort in the agents as we stated in the introduction of this paper. For instance, in our blocks world example, a value can be stated as "preserve structural stability". Agents A1 and A2 might engage in a dialogue when they are seeking the next block to put on, when one finds a block and tells the other "I found one", the second agent who does not see a suitable block might say "go ahead". The permission by the second agent bolsters the agent's autonomy in attempting to place the block. When they have to place blocks simultaneously, once both agents have found suitable blocks for placement, one of them might take lead and say "let's go". Taking lead is a function of experiencing a relatively higher autonomy. Elsewhere, we have discussed relative autonomy (Brainov and Hexmoor 2001, Hexmoor 2002a, 2002b, and 2001). In this example, this relative autonomy might be due to relative skills in block placement or a norm such as "whoever finds a block and waits for the second agent takes lead." At any rate, taking lead implies control. As we explained earlier, the follower observes the actions of the lead and matches its actions for coordinated effort. If the follower can't keep up, it might say "wait" indicating a request for help and setting up an indirect adjustment in autonomy and control.

So far we have discussed how social relationships are interdependent and can be used to coordinate and produce a coherent set of actions. Social actions of one agent might affect another who might repeat by a social action that produces an influence on another and so on. There might also be shared or joint social influences. For example if the structure of blocks become unstable, both agents might independently experience an obligation to protect the structure. But when they become aware of one another's obligation, they arrive at a joint responsibility to protect the structure. This joint influence will produce a joint autonomy, which might lead to a joint action such as steadying the table under the structure. This is the basis of group influence and group action.[3]

We said that social influences rely on prevailing norms and values. An agent might have several levels of such values and norms at any one time. Obligations to uphold the ultimate group intent are derived from corresponding values and norms at the global level. These obligations may compete against an agent's social influences at lower levels. Let's differentiate values and norms for an agent into n levels with level 1 being the highest (i.e., ultimate) and levels n being the lowest. Values and norms will be labeled with their level as values V_i and norms N_i. Let's redefine function f with f^i that maps an agent's set of beliefs B, a set of values V_i, a set of norms N_i, and a set of social actions A to a set of influences I^i:

$$f^i: B \times V_i \times N_i \times A \rightarrow I^i.$$

As designers of agent systems we can design mechanisms for encoding the desired ontological level to match the agent's responsibility level. This design-time responsibil-

[3] See http://csce.uark.edu/~hexmoor/AAAI-02/AAAI-02-cfp.htm for a workshop on this topic.

ity encoding is a method that can be used to assure predictable agent behavior. If we design obligation categories (i.e., responsibilities within ontological levels for the agent), an agent might be directed to adopt specific obligations about certain tasks to perform on behalf of a chosen agent or the human user in case the agent interacts with a human. This will affect the agent's autonomy and control with respect to the agent (or the user). For example if the project is safety-critical, overall project goals (and corresponding values and norms) are given a higher ontological status in the agent's makeup.

Agents might experience simultaneous social actions that have influences at different value and norm levels. These influences will also have different magnitudes. At times there might be conflicts among these influences. The conflict might be within an agent or between agents. A simplistic conflict resolution for an agent is the rule "if an influence at a high level is conflicted with an influence at a low level and as long as the magnitude of the influence at the low level is not much larger than the one at the high level, choose the influence at the high level." I.e., higher influences suppress the lower influences unless the strength of influence at the low level is significantly larger than the influence at the high level. In the exception case, we can introduce case-by-case domain rules to make sure hierarchies are largely maintained but specific over-riders are possible.

When agents share ontological levels of values and norms, it is easy to see that they have a greater chance of harmony. Conflicts among such agents can also be resolved using our resolution rule. In this case, one agent might sacrifice its highest social influence for another's even higher social influence.

Overall missions can be guaranteed among agents who share the values pertaining to that mission if we specify certain social influence tolerances in agents. First, we can specify how much tolerance we allow for adverse social influences before reacting to them. Next, we can specify the threshold of deviation from other social influences for suppression of lower level social influences.[4] We can use this method to other levels of norms and values and produce similar guarantees at those levels. The notion of guarantee we introduce here differs from validation and verification. In validation and verification, programs are tested to obey certain properties (Engelfriet, et al 2002). This is not easily possible with multiagent programs that have many more paths of execution due to the level of autonomy we provide agents. For instance, in nontrivial systems, chain effects of social influences are too complex to account for agent actions. We envision methods for setting up combinations of norms and values such that they contain unfolding chains and suppress undesirable results of influence chains.

Consider our blocks world example with three agents instead of two. As a delegation social action, all three agents might have found blocks that can fit in block positions 3 and 4 and out of politeness (and the corresponding social influence) one agent might say to another "go ahead" (this is a social action) and second agents might experience the same influence and say "go ahead" to the third agent and back to the first agent. A norm that can break this influence chain is "if politeness leads to inaction, the earliest agent to be polite will proceed".

4 Conclusions

We outlined how social actions generate social influences and showed a few salient interdependencies among social influences. We then discussed agents that can be designed to favor social influences that pertain to their highest level of norms and values and some exceptions. If agents shared norms and values, we can design agents that guarantee guarding against adverse social influences, suppression of social influences due to lower level norms and values, and undesirable chains of influence. This gives us a practical methodology for using social influence in implementing social responsibility among benevolent agents.

Acknowledgements

This work is supported by AFOSR grant F49620-00-1-0302.

[4] Agents who share mission level values and norms might not share norms and values at lower levels.

References

S. Brainov and H. Hexmoor, 2001. Quantifying Relative Autonomy, In *Multiagent Interaction, In IJCAI-01 Workshop, Autonomy, Delegation, and Control.*

S.S. Brehm S.M. Kassin, S. Fein 2002. *Social Psychology*, Houghton Mifflin pub.

J. Engelfriet, C.M. Jonker, and J. Treur, (In press 2002). Compositional Verification of Multi-Agent Systems, In Temporal Multi-Epistemic Logic, *Journal of Logic, Language and Information.*

H. Hexmoor, (In Press, 2002a). In Search of Simple and Responsible Agents, In the Proceedings of *NASA GSFC/JPL Workshop on Radical Agents*, MD.

H. Hexmoor, (In Press, 2002b). From Inter-Agents to Groups, In *International Symposium in Artificial Intelligence, ISAI-01*, India.

H. Hexmoor, 2001. A Cognitive Model of Situated Autonomy, In Advances in Artificial Intelligence, Springer LNAI2112 -pages 325-334, Kowalczk, Wai Loke, Reed, and William (eds).

T. Ohguro. K. Kuwabara, T. Owada, and Y. Shirai, 2001. FaintPop: In touch with the social relationships, In International Workshop on Social Intelligence Design, The 15th Annual Conference of JSAI, Japan.

S. Kassin, 2001. *Psychology*, Third Edition, , Prentice-Hall.

C. U. Larson, 1998. *Persuasion: Reception and Responsibility,* 9th edition. Boston: Wadsworth.

M. Wooldridge, 2000. *Reasoning about Rational Agents*, MIT Press.

An Agent-Based Network Management System

D.N.Legge; P.R.Baxendale
Centre for Telecommunication Networks,
School of Engineering,
University of Durham
d.n.legge@dur.ac.uk; peter.baxendale@dur.ac.uk

Abstract

This paper describes ongoing research work to build a network management system from autonomous software agents, using the BT Zeus Agent Building Toolkit. The network specific to this work is a 6-Node ATM Testbed, which is fully reconfigurable; however, it is intended that the management system is generic enough to be implemented on different networks. Each node of the network hosts its own agent society; within which the role of each agent has an useful analogy to the controlling of the layers of the ISO OSI reference model. Although the lower-layer agent's actions must be tightly prescribed, it is intended that higher-layer agents will take on more strategic, longer-term outlooks, and be more adaptive and autonomous. Agent societies are able to communicate, to allow their collaboration; it is intended that this will only be at the peer-to-peer level. The long-term aim of this project is to implement fully distributed control within a network, while maintaining the ability to take more holistically informed decisions for network-wide and longer-term strategies.

1 Introduction

A classical paradox in the operation of telecommunications networks is the centralisation or distribution of control. A network is said to have centralised control when one node takes all the decisions. Here there is a clear leader, and the assumption is that it can make impartial and co-ordinated decisions. Problems arise when the network is geographically distributed, and the central node has to make decisions with an incomplete and possibly out-dated knowledge. Also, a link failure could cause the isolation of part of the network, or if the central node itself fails then the whole network could become inoperable.

In the other extreme, each node makes all of its own decisions. As a whole, the Internet runs on this basis; and it can't be denied that it is highly resilient, and for non-critical data is perfectly adequate. However, the Internet provides little if anything in the way of Quality of Service (QoS) guarantees. The whole of the Internet provides a "best-effort" in terms of QoS, and this is inherent in the protocols developed, certainly until the new Internet Protocol (IP), version 6. ATM (Asynchronous Transfer Mode) however, is a protocol designed from first principles to enable guarantees on the QoS to be provided. It achieves this by tightly controlling its resources, agreeing contracts with users before admission to the network. It can then police those contracts continuously.

Jennings [1999] defines agents succinctly as "situated problem solvers" and as such are ideal for application to the complex situations that arise in telecommunications network management; allowing network management software to be autonomous and adaptive reduces the need for human intervention and, given the high network speeds, can reduce downtime, loss of service and consequently revenue. The agents can then learn from situations and provide better solutions as it increases its knowledge base. This paper describes the first steps to build such a system.

A perceived problem with ATM is the complexity of its configuration. Therefore any automation of the process of configuration would be of benefit. At the same time, care must be taken to minimise the additional network traffic generated by the management system itself (referred to as Control Traffic).

2 Current Work

There is a number of areas in which agents could be utilised within telecommunications, and indeed there have been implementations; some of them real systems, while some are simulated.

Hayzelden [1999], and indeed the whole ACTS-IMPACTS[1] project, utilises agents to implement Connection Admission Control (CAC). This procedure is part of the ATM protocol standard - put simply, the agents decide whether the network will allow a user application to send a particular set of traffic onto the network.

Gaïti [1996a] and [1996b] describe agents being used to detect congestion in the network, and dynamically change the thresholds so that the congestion control mechanisms can be triggered by higher or lower network load levels.

Another area of agents being used in telecommunications control at the simulation stage, is the application of ant-based techniques to routing problems, such as Vittori [2001].

3 Aim

It is proposed that by employing a society of autonomous software agents on each node, an effective Network Management system can be developed. To provide a structure to this society, it is proposed that individual agents could be mapped to a hierarchy analogous to the a standard protocol model in the telecommunications world – the ISO OSI[2] model; discussion of this model is beyond the scope of this paper, but details can be found in any standard telecommunications text, e.g. [Walrand 1999]. This model provides defined interfaces and levels of abstraction to enable communication protocols to be developed which are interoperable with a generic higher and lower layer; effectively introducing platform independence.

Each layer descending the stack considers increasingly local issues – and it is this property that should allow fully distributed control, while maintaining the ability to communicate and produce network-wide, strategies – which are impossible where no such communications exist. The layering of the agents also mean that each agent need only be aware of the agent layer directly above and below, simplifying the interactions (although there will be a need for "utility agents" to provide services which 'glue' the whole society together).

An alternative analogy is that of speed; an agent controlling a switch will need to be almost dedicated to the task. A layered structure will help to gradate between these fast, reactive agents nearest to the hardware, up to more proactive, slower agents at the higher layers. These higher layers will enable more adaptive behaviour, according to long-term trends, while monitoring the actions of the lower layer agents; this is similar to the Subsumption Architecture discussed by [Hayzelden 1999].
A further aim is to show the feasibility of producing such a management system using standard-compliant, readily available tools.

4 Tools

4.1 ATM Testbed

The School of Engineering owns a six-node ATM network, dedicated to research work. This is shown in Figure 1 with all possible connections made. This means that any topology can be configured and any test scenario used. All six switches, two UNIX machines and six PCs are connected by ethernet connections on a firewall protected network. Video codecs and network analysers can be used to generate and capture traffic.

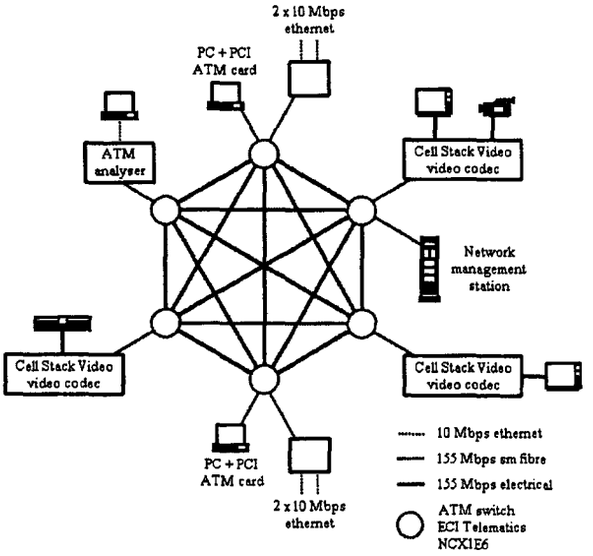


Figure 1: Diagram of Testbed

4.2 BT's Zeus Agent Building Toolkit

In searching for a toolkit to aid the building of the agent society, a great many were discovered. Most of these were developed for a specific research project and as such were not adequately documented or supported. However the BT Zeus agent toolkit seems not to suffer from these difficulties. It is also written in Java, which is useful given the mix of operating systems and machines available, and the platform independence it provides.

[1] http://www.acts-impact.org/
[2] ISO OSI - International Standards Organisation, Open Systems Integration

4.3 Java Virtual Machines

Because of the difficulty of installing a Java Virtual Machine on the ATM switches within the testbed network, the PCs are used as platforms to run the agents. A low level agent then controls the switches remotely. This does not affect the functionality, as the agents can control all aspects of the switches from the PCs, but it does allow the use of a friendlier front end and Graphical User Interface.

5 Design

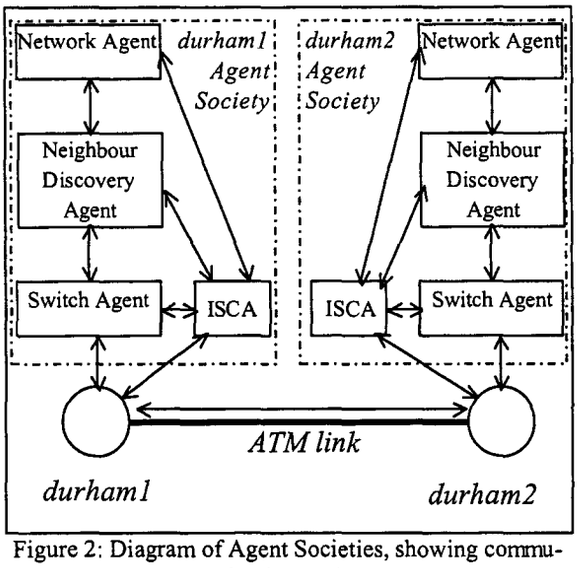


Figure 2: Diagram of Agent Societies, showing communication paths.

As discussed above, the OSI model provides a useful parallel for determining the scope of each agent in the society. This model is layered to provide increasing levels of abstraction from the physical interface. The bottom three layers - the Physical layer, the Link layer and the Network layer - are currently represented in our structure by one agent each. The first stage in this research project has been to implement these. In designing the agents, it was found useful to use the "Sphere of Responsibility" (SoR) test, described by [Collis 2000]. The current agent society is shown in Figure 2, while the roles of these agents are discussed in the next section.

5.1 Outline of Agents

5.1.1 Switch Control Agent (Physical Layer)

The Switch Control Agent provides (remotely in this case) access to the hardware resources in the switch. This enables both the discovery and configuration of switch resources, including, for example, recognising the presence of links. The agent keeps a record of what physical interfaces are present, if they have been initialised, and a reference to the control channel used for communication on that interface – which will be used for communication with the node at the other end of that link. When the agent has established communication with the switch, a message is sent to the Inter-Society-Communication Agent (ISCA, see below) to start listening for incoming messages to the agent society.

5.1.2 Neighbour Discovery Agent (Link Layer)

The first task that the agent society must do is to discover the immediate connections to the host node. It was decided that a single agent on each node should handle this, and store the results. The information stored about the link by this agent is the destination node and the local port numbers at either end.
The agent has a refresh function, in case other nodes come online at a later point, this is manually triggered currently, but should be an automatic function depending on a timeout; however, it is this timeout that will greatly affect the amount of Control Traffic generated.

5.1.3 Network Agent (Network Layer)

When the immediate neighbours are known, the next stage is to attempt to discover any other nodes on the network. This is done by exchanging knowledge with the nodes directly connected. In this way, knowledge of the network ripples across the network; if the agents are all refreshing at the same rate, each refresh will spread knowledge of a node by one hop. This raises the question of scalability, however without a hierarchical network structure this is inevitable.

This process effectively compiles a routing table. The agent can then make decisions on the best route to get to a particular node. The information stored in the routing table is the name of the destination node, the local port number to use and some representation of the distance to it; known as a "metric".

What is used as the metric is the source for much debate, and beyond the scope of this paper. At its simplest - as in this case - it can be merely the number of "hops" (The number of nodes traversed). However, it could be any quantifiable and comparable parameter to discriminate between a number routes. Other parameters used include: present load across the link, reserved bandwidth, available bandwidth, areas of congestion on the network, historical popularity of links and monetary factors. As could be imagined, the weighting of all these factors will be a real challenge. Eventually the agent could dynamically change them depending on circumstances. It is an area where one could lose sight of the real goal of usability in calculating accurate values for the metric.

The agent is designed to store a primary route and a back-up route in case of failure; the primary route having the lowest metric. Having a back-up route offers a balance between resilience in the event of failure, against the flooding of the network with control traffic in finding every possible route between two points. The timeout period for refreshing is another finely balanced decision, and is discussed in Shaikh [2001] but currently is user prompted. When the agent is given control of this timing a much greater level of autonomy will have been achieved, according to the definitions of agentism in [Wooldridge 1997] and [Gaïti 1996b] and this is certainly intended.

Presently only peer-to-peer communication is allowed between societies; and there seems no reason to break this convention, as this simplifies the communication. This means that communication can only occur vertically - up and down the "stack" within the same society - or horizontally - to the same layer in another society. The ability to communicate with other agent societies (on adjoining nodes) requires a further agent, described next.

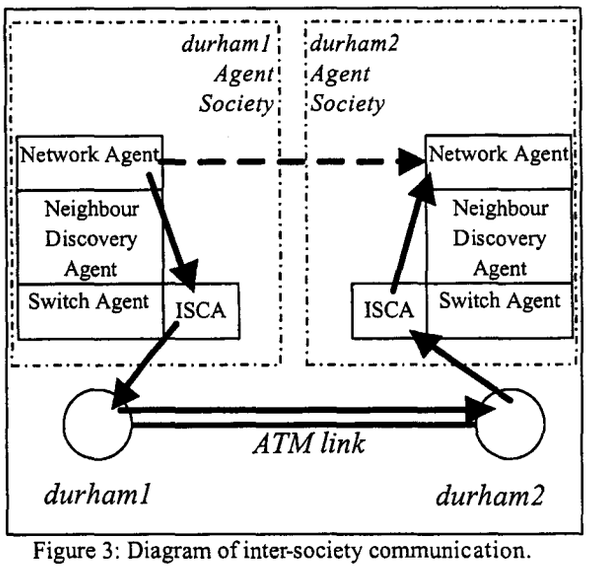


Figure 3: Diagram of inter-society communication.

5.1.4 Inter-Society Communication Agent, ISCA

This agent does not represent a layer of the ISO model, as it is really a utility agent. Similarly to the Switch Agent, it maintains a connection to the ATM switch, across which messages to other societies are directed, and then forwarded by the switch on the appropriate link.

Although an ethernet connection is available it was felt that using this for inter-society communication was not in the spirit of the research, so the ATM links are used. Each agent across the network is uniquely identifiable by its name and the society it belongs to; thus giving it an address such as network@durham1. This is analogous to a normal email address. Figure 3 shows the path of a message from *network@durham1* to *network@durham2*.

The use of the ATM links for inter-society communications also allows us to measure the aggregate Control Traffic using an ATM traffic analyser. It is also important to keep the communications to within existing protocols; by employing a standard-compliant Agent Communication Language (ACL) and not relying on a proprietary implementation - this is aided by the use of an off-the-shelf toolkit.

6 Results

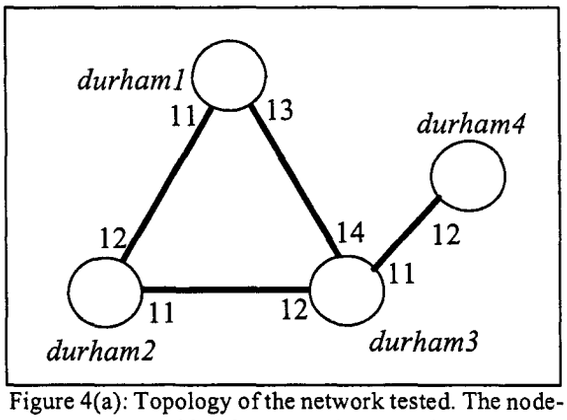


Figure 4(a): Topology of the network tested. The node-names and local port numbers are shown.

node	port	metric	altport	altmetric
durham2	11	1	13	2
durham3	13	1	11	2
durham4	13	2	11	3

Figure 4(b): Output of Network Agent on *durham1*

node	port	metric	altport	altmetric
durham1	12	1	11	2
durham3	11	1	12	2
durham4	11	2	12	3

Figure 4(c): Output of Network Agent on *durham2*

node	port	metric	altport	altmetric
durham1	14	1	12	1
durham2	12	1	14	2
durham4	11	1	0	100

Figure 4(d): Output of Network Agent on *durham3*

node	port	metric	altport	altmetric
durham1	12	2	0	100
durham2	12	2	0	100
durham3	12	1	0	100

Figure 4(e): Output of Network Agent on *durham4*

Currently, the four agents described above have been implemented, and successfully tested. As a useful analogy, these map to the first three layers of the OSI model. The lowest layer (the Switch Agent) establishes what ports are available, and at the request of the second layer (the Neighbour Discovery Agent) establishes the identity of nodes at the other end of each link. The Link Agent can then be questioned by the Network Agent at the third layer which then establishes the existence of other nodes/societies both on direct links and by questioning these, further afield.

In doing this, the Network Agent at each node discovers the network topology, is tasked to produce a routing table, so performing some of the configuration automatically. Figure 4 shows this being achieved over four sparsely-connected nodes (i.e. not fully-connected). Figure 4(a) shows the network topology, while 4(b), (c), (d) and (e) show the routing table produced at each node. As can be seen each node learns of all the nodes on the network, not just those that it is directly connected to. It also discovers a backup route (prefixed with "alt"); although this can share links with the primary route, they do diverge somewhere. Where the metric is "100", no back-up route has been discovered.

7 Further Work

Although the work described above is a working system, the intended functionality extends considerably further than currently implemented. The aim of the project is to harness the advantages of agent-based software to control a telecommunication network. These advantages include the ability to work autonomously, adapt to their environment and learn from situations that arise.

7.1 Further Agents

There are currently plans for a number of other agents; these include one to manage the situation of a failure in the network, and one to handle the mapping of IP over ATM. These are described next.

7.1.1 Failure Recovery Agent

An interesting situation is presented when a node or link fails. In this case rather than allowing knowledge of the failure to ripple through the network, it will be necessary to notify nodes explicitly. There are two possible situations here:

Node Failure: If a node fails then the routing tables need to be amended, removing the node. Any traffic with that node as its destination will require the source to be notified. Note that the failure of a node could be seen as being equivalent to multiple link failure.

Link Failure: In the event of link failure, the traffic must be routed round the problem. Depending on the topology of the network, a link failure could isolate part of the network, making it equivalent to a node failure.

Whichever situation it is, traffic will have to be re-routed. Initially, the furthest downstream node (i.e. next to the failure) must consult its routing table, and attempt to set up connections for the traffic around the problem. If the node can't find an alternative route, the responsibility passes back to the previous node - which will try itself. Cases where traffic ends up doubling back on itself need to be detected, and corrected.

This re-routing is a similar function to a normal routing algorithm, but there is the additional problem of dealing with traffic flows already using the network, and so having an existing traffic contract. Although a failure could be seen as a *force majeure,* the network should try to re-route if at all possible.

To achieve all of this, the Failure Recovery agent will have to propagate failure messages to update routing tables. It must then handle the attempts at re-routing. The failure of a major link could have large knock-on affects across the network. The agents across the network will have to adapt to the new conditions, which will possibly affect many routes.

7.1.2 DiffServ Agent

Originally, it was intended that ATM would be *the* ubiquitous protocol, going straight to the desktop PC. However, with the explosion of the Internet[3], ATM has found its niche at the backbone level of networks (i.e. high speed trunk routes).

As stated above, the Internet has very little in the way of QoS provision. There are attempts to patch some in however. One of them, the most likely to succeed, is known as Differentiated Services, or DiffServ[4] [Blake 1998].

DiffServ associates IP packets with a number of traffic classes using "code points". The code point is identified in the IP header and the mapping of this to a traffic class and therefore QoS requirements is domain dependant. The code point aggregates traffic with similar QoS requirements. At the boundary of an ATM core network designed to carry IP traffic, decisions must be made on how to map DiffServ traffic to ATM resource requirements and hence routes.

[3] Which is one *very* large TCP/IP network.
4 Also referred to as DS.

The use of agents would allow this process to adapt to changing traffic patterns and resource availability within the core network.

8 Conclusions

So far, a system has been developed that automatically establishes all the connections available at a node, and then compiles a routing table of all nodes it can learn about. This can only be achieved this by co-operating with the surrounding nodes. This is a first basic step towards network configuration and provides the framework for a more adaptive and dynamic network control system. As this is ongoing research a number of other agents are planned, including more proactive, strategic-planning agents.

As already stated, the aim of this project is to build a fully operational network control system capable of running autonomously, and able to adapt to changing circumstances. An emphasis is placed on achieving this through using off-the-shelf components, to ensure compliance to general agent standards; but most specifically in the area of agent communication, which is the greatest empowering property of an agent.

Acknowledgements

This work is being carried out under an EPSRC grant. The system built was produced using the BT ZEUS Agent Building Toolkit, which is available from http://www.btexact.com/projects/agents.htm.

References

Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z. and Weiss W., "An Architecture for Differentiated Services", RFC 2475, IETF December 1998

Collis J. & Ndumu D. "The Zeus Agent Building Toolkit: The Role Modelling Guide" Release 1.02, BT plc. 2000

Gaïti D. & Boukhatem N. "Cooperative Congestion Control Schemes in ATM Networks", IEEE Communications Magazine, p102, November 1996a

Gaïti D. & Pujolle G. "Performance Management Issues in ATM Networks and Congestion Control", IEEE/ACM Transactions on Networking, 4(2):249-257, April 1996b

Hayzelden A.L.G., "A Multiple-Agent Approach for Resource Configuration in Communication Networks", PhD Thesis, Queen Mary and Westfield College, London, 1999

Jennings N.R., "Agent-based Computing: Promises and Perils" Proc. 16th Int. Joint Conf. on Artificial Intelligence (IJCAI-99), Stockholm, Sweden. 1429-1436, 1999.

Shaikh A., Rexford J., & Shin K.G., "Evaluating the Impact of Stale Link State on Quality-of-Service Routing", IEEE/ACM Transactions on Networking, 9(2):162-175, April 2001

Vittori K., & Araújo F.R., "Agent-Oriented Routing in Telecommunications Networks", IEICE Transactions on Communications, E84-B(11):3006-3013, November 2001

Walrand J. & Varaiya P., "High-Performance Communication Networks", 2nd Ed, Morgan Kaufmann, 1999

Wooldridge M. "Agent-Based Software Engineering" IEE Proceedings in Software Engineering 1997