AISB Journal

The Interdisciplinary Journal of
Artificial Intelligence and the Simulation of Behaviour

Volume 1 – Number 2 – June 2002

The Journal of the
Society for the Study of Artificial Intelligence
and the Simulation of Behaviour
http://www.aisb.org.uk

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

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

ISSN 1476-3036

Contents

AISB Journal **1**(2), © 2002
http://www.aisb.org.uk

Finding Counterexamples to Inductive Conjectures and Attacking Security Protocols

Graham Steel, Alan Bundy and Ewen Denney

Division of Informatics, University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, Scotland
G.J.Steel@ed.ac.uk ; A.Bundy@ed.ac.uk ; Ewen.Denney@ed.ac.uk

Abstract

We present an implementation of a method for finding counterexamples to universally quantified inductive conjectures in first-order logic. Our method uses the proof by consistency strategy to guide a search for a counterexample and a standard first-order theorem prover to perform a concurrent check for inconsistency. We explain briefly the theory behind the method, describe our implementation, and evaluate results achieved on a variety of incorrect conjectures from various sources.

Some work in progress is also presented: we are applying the method to the verification of cryptographic security protocols. In this context, a counterexample to a security property can indicate an attack on the protocol, and our method extracts the trace of messages exchanged in order to effect the attack. This application demonstrates the advantages of the method, in that quite complex side conditions decide whether a particular sequence of messages is possible. Using a theorem prover provides a natural way of dealing with this. Some early results are presented and we discuss future work.

1 Introduction

Inductive theorem provers are frequently employed in the verification of programs, algorithms and protocols. However, programs and algorithms often contain bugs, and protocols may be flawed, causing the proof attempt to fail. It can be hard to interpret a failed proof attempt: it may be that some additional lemmas need to be proved or a generalisation made. In this situation, a tool which can not only detect an incorrect conjecture, but also supply a counterexample in order to allow the user to identify the bug or flaw, is potentially very valuable. Incorrect conjectures also arise in automatic inductive theorem provers where generalisations are speculated by the system. Often we encounter the problem of over-generalisation: the speculated formula is not a theorem. A method for detecting these over-generalisations is required.

Proof by consistency is a technique for automating inductive proofs in first-order logic. Originally developed to prove correct theorems, this technique has the property of being refutation complete, i.e. it is able to refute in finite time conjectures which are inconsistent with the set of hypotheses. When originally proposed, the technique was of

limited applicability. Recently, Comon and Nieuwenhuis have drawn together and extended previous research to show how it may be more generally applied, (Comon and Nieuwenhuis, 2000). They describe an experimental implementation of the inductive completion part of the system. However, the check for refutation or consistency was not implemented. This check is necessary in order to ensure a theorem is correct, and to automatically refute an incorrect conjecture. We have implemented a novel system integrating Comon and Nieuwenhuis' experimental prover with a concurrent check for inconsistency. By carrying out the check in parallel, we are able to refute incorrect conjectures in cases where the inductive completion process fails to terminate. The parallel processes communicate via sockets using Linda, (Carreiro and Gelernter, 1989).

The ability of the technique to prove complex inductive theorems is as yet unproven. That does not concern us here – we are concerned to show that it provides an efficient and effective method for refuting *incorrect* conjectures. However, the ability to prove at least many small theorems helps alleviate a problem reported in Protzen's work on disproving conjectures, (Protzen, 1992) – that the system terminates only at its depth limit in the case of a small unsatisfiable formula, leaving the user or proving system none the wiser.

The system has been tested on a number of examples from various sources including Protzen's work, (Protzen, 1992), Reif et al.'s, (Reif et al., 2000), and our own work. We also have some early results from our work in progress, which is to apply the technique to the problem of cryptographic security protocol verification. These protocols often have subtle flaws in them that are not detected for years after they have been proposed. By devising a first-order version of Paulson's inductive formalism for the problem, (Paulson, 1998), and applying our refutation system, we can not only detect flaws but also automatically generate the sequence of messages needed to expose these flaws. By using an inductive model with arbitrary numbers of agents and runs rather than the finite models used in most model-checking methods, we have the potential to synthesise parallel session and replay attacks where a single principal may be required to play multiple roles in the exchange.

In the rest of the paper, we first review the literature related to the refutation of incorrect conjectures and proof by consistency, then we briefly examine the Comon-Nieuwenhuis method. We describe the operation of the system, relating it to the theory, and present and evaluate the results obtained so far. Our work in progress on the application of the system to the cryptographic protocol problem is then presented. Finally, we describe some possible further work and draw some conclusions.

2 Literature Review

2.1 Refuting Incorrect Conjectures

At the CADE-15 workshop on proof by mathematical induction, it was agreed that the community should address the issue of dealing with non-theorems as well as theorems [1]. However, relatively little work on the problem has since appeared. Previously, in the early nineties, Protzen presented a sound and complete calculus for the refutation of faulty conjectures in theories with free constructors and complete recursive definitions, (Protzen, 1992). Given a (possibly) incorrect conjecture, the method of refutation is as follows: first, formulate an existentially quantified negation of the conjecture. Then, attempt to

[1] The minutes of the discussion are available from `http://www.cee.hw.ac.uk/~air/cade15/cade-15-mind-ws-session-3.html`.

find an instantiation of the existentially quantified variables such that the counterexample statement is satisfied. The search for the counterexample is guided by the recursive definitions of the function symbols in the conjecture. A depth limit ensures termination when no counterexample can be found. Protzen implemented the counterexample finder as part of the INKA inductive theorem prover, (Hutter and Sengler, 1996). Its chief application in this context is to refute incorrect generalisations.

More recently, Reif et al., (Reif et al., 2001), have implemented a method for counterexample construction that is integrated with the interactive theorem prover KIV, (Reif, W., 1995). Their method incrementally instantiates a formula with constructor terms and evaluates the formulae produced using the simplifier rules made available to the system during proof attempts. A heuristic strategy guides the search through the resulting subgoals for one that can be reduced to $false$. If such a subgoal is not found, the search terminates when all variables of generated sorts have been instantiated to constructor terms. In this case, a model condition remains, which must be used by the user to decide whether the instantiation found is a valid counterexample.

2.2 Proof by Consistency

Proof by consistency is a technique for automating inductive proof. It has also been called *inductionless induction*, and *implicit induction*, as the actual induction rule used is described implicitly inside a proof of the conjecture's consistency with the set of hypotheses. Recent versions of the technique have been shown to be *refutation complete*, i.e. are guaranteed to detect non-theorems in finite time.[2] The proof by consistency technique was developed to solve problems in equational theories, involving a set of equations defining the *initial model*[3], E. The first version of the technique was proposed by Musser, (Musser, 1980), for equational theories with a *completely defined equality predicate*, i.e. a predicate eq with the property that two terms are equal under the congruence relation given by the equation set E if and only if $eq(s,t) = true$, and unequal if and only if $eq(s,t) = false$. This requirement placed a strong restriction on the applicability of the method. The completion process used to deduce consistency was the Knuth-Bendix algorithm, (Knuth and Bendix, 1970). However, the KB process does not always terminate, and it fails if the conjecture is unorientable (e.g. commutativity).

Huet and Hullot, (1982), extended the method to theories with free constructors, and Jouannaud and Kounalis, (1989), extended it further, requiring that E should be a convergent rewrite system. Bachmair, (1991), proposed the first refutationally complete deduction system for the problem, using a *linear strategy* for inductive completion. This is a restriction of the Knuth-Bendix algorithm which entails only examining overlaps between axioms and conjectures. The key advantage of the restricted completion procedure was its ability to cope with unoriented equations. The refutational completeness of the procedure was a direct result of this.

The technique has been extended to the non-equational case. Ganzinger and Stuber, (1992), proposed a method for proving consistency for a set of first-order clauses with equality using a refutation complete linear system. Kounalis and Rusinowitch, (1990), proposed an extension to conditional theories, laying the foundations for the method implemented in the SPIKE theorem, (Bouhoula and Rusinowitch, 1995). Ideas from the

[2]Such a technique must necessarily be incomplete with respect to proving theorems correct, by Gödel's incompleteness theorem.

[3]The initial or standard model is the minimal Herbrand model. This is unique in the case of a purely equational specification.

proof by consistency technique have been used in other induction methods, such as cover set induction, (H. Zhang, 1988), and test set induction, (Bouhoula et al., 1992).

3 Theory of the Comon-Nieuwenhuis Method

Comon and Nieuwenhuis, (Comon and Nieuwenhuis, 2000), have shown that the previous techniques for proof by consistency can be generalised to the production of a first-order axiomatisation $\mathcal{A}$ of the minimal Herbrand model such that $\mathcal{A} \cup E \cup C$ is consistent if and only if C is an inductive consequence of E. With $\mathcal{A}$ satisfying the properties they define as an *I-Axiomatisation*, inductive proofs can be reduced to first-order consistency problems and so can be solved by any saturation based theorem prover. We give a very brief summary of their results here. Suppose I is, in the case of Horn or equational theories, the unique minimal Herbrand model, or in the case of non-Horn theories, the so-called *perfect model* with respect to a total ordering on terms, $\succ$:

Definition 1 *A set of first-order formulae $\mathcal{A}$ is an* I-Axiomatisation *of I if*

1. *$\mathcal{A}$ is a set of purely universally quantified formulae*
2. *I is the only Herbrand model of $E \cup \mathcal{A}$ up to isomorphism.*

An I-Axiomatisation is normal *if $\mathcal{A} \models s \neq t$ for all pairs of distinct normal terms s and t*

The I-Axiomatisation approach produces a clean separation between the parts of the system concerned with inductive completion and inconsistency detection. Completion is carried out by a saturation based theorem prover, with inference steps restricted to those produced by conjecture superposition, a restriction of the standard superposition rule. Each non-redundant clause derived is checked for consistency against the I-Axiomatisation. If the theorem prover terminates with saturation, the set of formulae produced comprise a *fair induction derivation*. The key result of the theory is this:

Theorem 1 *Let $\mathcal{A}$ be a normal I-Axiomatisation, and $C_0, C_1, \ldots$ be a fair induction derivation. Then $I \models C_0$ iff $\mathcal{A} \cup \{c\}$ is consistent for all clauses c in $\bigcup_i C_i$.*

This theorem is proved in (Comon and Nieuwenhuis, 2000). Comon and Nieuwenhuis have shown that this conception of proof by consistency generalises and extends earlier approaches. An equality predicate as defined by Musser, a set of free constructors as proposed by Huet and Hullot or a ground reducibility predicate as defined by Jouannaud and Kounalis could all be used to form a suitable I-Axiomatisation. The technique is also extended beyond ground convergent specifications (equivalent to saturated specifications for first-order clauses) as required in (Jouannaud and Kounalis, 1989; Bachmair, 1991; Ganzinger and Stuber, 1992). Previous methods, e.g. (Bouhoula and Rusinowitch, 1995), have relaxed this condition by using conditional equations. However a ground convergent rewrite system was still required for deducing inconsistency. Using the I-Axiomatisation method, conjectures can be proved or refuted in (possibly non-free) constructor theories which cannot be specified by a convergent rewrite system.

Whether these extensions to the theory allow larger theorems to be *proved* remains to be seen, and is not of interest to us here. We are interested in how the wider applicability of the method can allow us to investigate the ability of the proof by consistency technique to root out a counterexample to realistic *incorrect* conjectures.

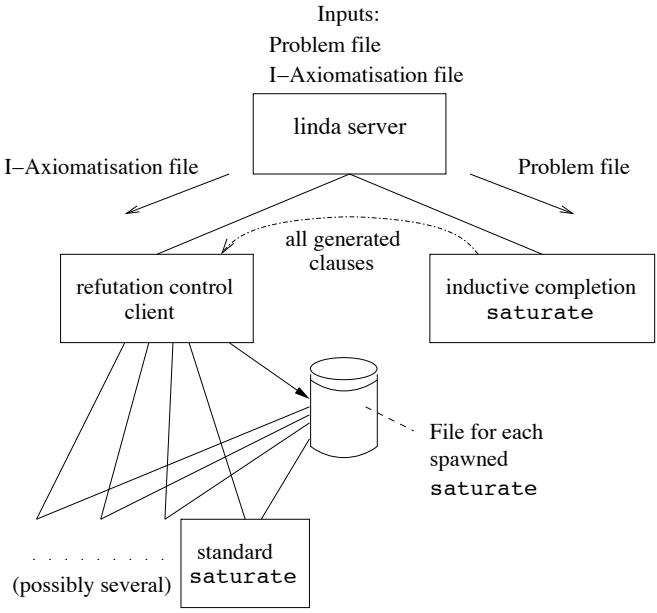


Figure 1: System operation

4 Implementation

The diagram in figure 1 illustrates the operation of our system. The input is an inductive problem in `Saturate` format and a normal I-Axiomatisation (see Definition 1, above). The version of `Saturate` customised by Nieuwenhuis for implicit induction (the right hand box in the diagram) gets the problem file only, and proceeds to pursue inductive completion, i.e. to derive a fair induction derivation. Every non-redundant clause generated is passed via the server to the refutation control program (the leftmost box). For every new clause received, this program generates a problem file containing the I-Axiomatisation and the new clause, and spawns a standard version of `Saturate` to check the consistency of the file. Crucially, these spawned `Saturate`s are not given the original axioms – only the I-Axioms are required, by Theorem 1. This means that almost all of the search for an inconsistency is done by the prover designed for inductive problems and the spawned `Saturate`s are just used to check for inconsistencies between the new clauses and the I-Axiomatisation. This should lead to a false conjecture being refuted after fewer inference steps have been attempted than if the conjecture had been given to a standard first-order prover together with all the axioms and I-Axioms. We evaluate this in the next section.

If, at any time, a refutation is found by a spawned prover, the proof is written to a file and the completion process and all the other spawned `Saturate` processes are killed. If completion is reached by the induction prover, this is communicated to the refutation control program, which will then wait for the results from the spawned processes. If they all terminate with saturation, then there are no inconsistencies, and so the theorem has been proved (by Theorem 1).

There are several advantages to the parallel architecture we have employed. One is that it allows us to refute incorrect conjectures even if the inductive completion pro-

cess does not terminate. This would also be possible by modifying the main induction `Saturate` to check each clause in turn, but this would result in a rather messy and unwieldy program. Another advantage is that we are able to easily devote a machine solely to inductive completion in the case of harder problems. It is also very convenient when testing a new model to be able to just look at the operation of the inductive completion process before adding the consistency check later on, and we preserve the attractive separation in the theory between the deduction and the consistency checking processes.

A disadvantage of our implementation is that launching a new `Saturate` process to check each clause against the I-Axiomatisation generates some overheads in terms of disk access etc. In our next implementation, when a spawned prover reaches saturation (i.e. no inconsistencies), it will clear its database and ask the refutation control client for another clause to check, using the existing sockets mechanism. This will cut down the amount of memory and disk access required. A further way to reduce the consistency checking burden is to take of advantage of knowledge about the structure of the I-Axiomatisation for simple cases. For example, in the case of a free constructor specification, the I-Axiomatisation will consist of clauses specifying the inequality of non-identical constructor terms. Since it will include no rules referring to defined symbols, it is sufficient to limit the consistency check to generated clauses containing only constructors and variables.

5 Evaluation of Results

Table 1 shows a sample of results achieved so far. The first three examples are from Protzen's work, (Protzen, 1992), the next three from Reif et al.'s, (Reif et al., 2000), and the last three are from our work. The gcd example is included because previous methods of proof by consistency could not refute this conjecture. Comon and Nieuwenhuis showed how it could be tackled, (Comon and Nieuwenhuis, 2000), and here we confirm that their method works. The last two conjectures are about properties of security protocols. The 'impossibility property' states that no trace reaches the end of a protocol. Its refutation comprises the proof of a possibility property, which is the first thing proved about a newly modelled protocol in Paulson's method, (Paulson, 1998). The last result is the refutation of an authenticity property, indicating an attack on the protocol. This protocol is a simple example included in Clark's survey, (Clark and Jacob, 1997), for didactic purposes, but requires that one principal play both roles in a protocol run. More details are given in section 6.

Our results on Reif et al.'s examples do not require the user to verify a model condition, as the system described in their work does. Interestingly, the formula remaining as a model condition in their runs is often the same as the formula which gives rise to the inconsistency when checked against the I-Axiomatisation in our runs. This is because the KIV system stops when it derives a term containing just constructors and variables. In such a case, our I-Axiomatisation would consist of formulae designed to check validity of these terms. This suggests a way to automate the model condition check in the KIV system.

On comparing the number of clauses derived by our system and the number of clauses required by a standard first-order prover (SPASS), we can see that the proof by consistency strategy does indeed cut down on the number of inferences required. This is more evident in the larger examples. Also, the linear strategy allows us to cope with commutativity conjectures, like the third example, which cause a standard prover to go into a loop. We might ask: what elements of the proof by consistency technique are allowing us to

Problem	Counterexample found	No. of clauses derived to find refutation	No. of clauses derived by a standard prover
$\forall N, M. \neg(s(N) + M = s(0))$	$N = 0, M = 0$	2(+0)	2
$X \neq Y \wedge X \neq 0 \wedge Y \neq 0$ $\Rightarrow (X \geq Y \wedge Y \neq 0) \vee$ $(X \neq 0 \wedge Y = 0)$	$X = s(0)$, $Y = s(s(X))$	4 (+3)	6
$app(K, L) = app(L, K)$	$K = 0, L = s(X)$	9(+11)	> 15000
$sort(l_1) \wedge l_2 = ap(l_1, [head(l_3)])$ $\wedge length(l_3) \geq 2 * length(l_1)$ $\wedge l_3 \neq nil \wedge$ $member(head(l_1), tail(l_3))$ $\Rightarrow sort(l_2)$	$l_1 = [s(X)]$, $l_2 = [s(X), 0]$ $l_3 = [0, s(X)\|Y]$	55(+1)	76
All graphs are acyclic	$[e(a, a)]$	99(+1)	123
All loopless graphs are acyclic	$[e(s(X), a), e(a, s(X))]$	511(+1)	2577
$gcd(X, X) = 0$	$X = s(0)$	17(+2)	29
Impossibility property for Neuman-Stubblefield key exchange protocol	$[msg(1), msg(2),$ $msg(3), msg(4)]$	866 (+0)	1733
Authenticity property for simple protocol from (Clark and Jacob, 1997)	see section 6	730(+1)	3148

Table 1: Sample of results. *In the third column, the first number shows the number of clauses derived by the inductive completion prover, and the number in brackets indicates the number of clauses derived by the parallel checker to spot the inconsistency. The fourth column shows the number of clauses derived by an unmodified first-order prover when given the conjecture, axioms and I-Axioms all together.*

make this saving in required inferences? One is the refutation completeness result for the linear strategy, so we know we need only consider overlaps between conjectures and axioms. Additionally, separating the I-Axioms from the axioms reduces the number of overlaps between conjectures and axioms to be considered each time. We also use the results about inductively complete positions for theories with free constructors, (Comon and Nieuwenhuis, 2000). This was applied to all the examples except those in graph theory, where we used Reif's formalism and hence did not have free constructors. This is the probable reason why, on these two examples, our system did not make as large a saving in derived clauses.

The restriction to overlaps between conjectures and axioms is similar in nature to the so-called set of support strategy, using the conjecture as the initial supporting set. The restriction in our method is tighter, since we don't consider overlaps between formulae in the set of support. Using the set of support strategy with the standard prover on the examples in Table 1 produces a refutation with fewer clauses derived than the standard strategy. However, performance is still not as good as for our system, particularly in the free constructor cases. The set of support also doesn't fix the problem of divergence on unoriented conjectures, like the commutativity example.

The efficiency of the method in terms of clauses derived compared to a standard prover looks good. However, actual time taken by our system is much longer than that for the standard SPASS. This is because the `Saturate` prover is rather old, and was not designed to be a serious tool for large scale proving. As an example, the impossibility property took about 50 minutes to refute in `Saturate`, but about 40 seconds in SPASS, even though more than twice as many clauses had to be derived. We used `Saturate` for our first implementation as Nieuwenhuis had already implemented the proof by consistency strategy in the prover. A second implementation of the whole system using SPASS should give us even faster refutations, and is one of our next tasks.

Finally, we also tested the system on a number of small inductive theorems. Being able to prove small theorems allows us to attack a problem highlighted in Protzen's work: that if an candidate generalisation (say) is given to the counterexample finder and it returns a result saying that the depth limit was reached before a counterexample was found, the system is none the wiser as to whether the generalisation is worth pursuing. If we are able to prove at least small examples to be theorems, this will help alleviate the problem. Our results were generally good: 7 out of 8 examples we tried were proved, but one was missed. Comon and Nieuwenhuis intend to investigate the ability of the technique to prove more and larger theorems in future.

More details of the results including some sample runs and details of the small theorems proved can be found at `http://www.dai.ed.ac.uk/~grahams/linda`.

6 Application to Cryptographic Security Protocols

We now describe some work in progress on applying our technique to the cryptographic security protocol problem. Cryptographic protocols are used in distributed systems to allow agents to communicate securely. Assumed to be present in the system is a spy, who can see all the traffic in the network and may send malicious messages in order to try and impersonate users and gain access to secrets. A good introduction to the field is Clark's survey, (Clark and Jacob, 1997). One of the main thrusts of research has been to apply formal methods to the problem. Researchers have applied techniques from model checking, theorem proving and modal logics amongst others. Much attention is payed to the modelling of the abilities of the spy in these models. However, an additional considera-

tion is the abilities of the participants. Techniques assuming a finite model, with typically two agents playing distinct roles, rule out the possibility of discovering a certain kind of parallel session attack, in which one participant plays both roles in the protocol. The use of an inductive model allows us to discover these kind of attacks.

Paulson's inductive approach has been used to verify properties of several protocols, (Paulson, 1998). Protocols are formalised in typed higher-order logic as the set of all possible traces, a trace being a list of events like 'A sends message X to B'. This formalism is mechanised in the Isabelle/HOL interactive theorem prover. Properties of the security protocol can be proved by induction on traces. The model assumes an arbitrary number of agents, and any agent may take part in any number of concurrent protocol runs playing any role. Using this method, Paulson discovered a flaw in the simplified Otway-Rees shared key protocol, (Burrows et al., 1990), giving rise to a parallel session attack where a single participant plays both protocol roles. However, as Paulson observed, a failed proof state can be difficult to interpret in these circumstances. Even an expert user will be unsure as to whether it is the proof attempt or the conjecture which is at fault. By applying our counterexample finder to these problems, we can automatically detect and present attacks when they exist.

Paulson's formalism is in higher-order logic. However, no 'fundamentally' higher-order concepts are used – in particular there is no unification of higher-order objects. Objects have types, and sets and lists are used. All this can be modelled in first-order logic. The security protocol problem has been modelled in first-order logic before, e.g. by Weidenbach, (Weidenbach, 1999). This model assumed a two agent model with just one available nonce[4] and key, and so could not detect the kind of parallel session attacks described. Our model uses lists to represent message traces and sets to describe intruder knowledge, and allows an arbitrary number of agents to participate, playing either role, and using an arbitrary number of fresh nonces and keys.

In order to make the model amenable to the automatic discovery of attacks, we have tried to maintain simplicity whilst not sacrificing the required expressivity. The model is kept Horn by defining two-valued functions for checking the side conditions for a message to be sent, e.g. we define conditions for $member(X, L) = true$ and $member(X, L) = false$. This cuts down the branching rate of the search. The sets we use are specified in terms of the same operators used in Paulson's model, and the intruders abilities are identical. Giving the definitions in terms of the operators allows us to avoid giving a full first-order axiomatisation of arbitrary sets. An example illustrates some of these ideas. Here, we demonstrate the formalism of a very simple protocol included in Clark's survey to demonstrate parallel session attacks, (Clark and Jacob, 1997). Although simple, the attack does require principal A to play both roles. It assumes that A and B already share a secure key, K_{AB}. N_A denotes a nonce generated by A. Here is the protocol:

1. $A \rightarrow B : \{\!|\ N_A \|\!\}_{K_{AB}}$
2. $B \rightarrow A : \{\!|\ N_A + 1 \|\!\}_{K_{AB}}$

At the end of a run, A should now be assured of B's presence, and has accepted nonce N_A to identify authenticated messages. However, there is a simple parallel session attack (C_X represents the spy impersonating agent X):

1. $A \rightarrow C_B : \{\!|\ N_A \|\!\}_{K_{AB}}$

[4] A nonce is a unique identifying number.

2. $C_B \rightarrow A : \{\!|\ N_A |\!\}_{K_{AB}}$
3. $A \rightarrow C_B : \{\!|\ N_A + 1 |\!\}_{K_{AB}}$
4. $C_B \rightarrow A : \{\!|\ N_A + 1 |\!\}_{K_{AB}}$

At the end of the attack, A at least believes B is operational. B may be absent or may no longer exist. Our system has synthesised this attack automatically. Protocols in our model are defined as a series of rules defining what messages participants may send based on traffic they have seen in the network. Additionally, the spy may send anything he can at any time. The spies knowledge is based on what he can extract from all the traffic in the network so far. The rule defining message 1 looks like this:

$$\begin{aligned}&m(XT) = true \wedge agent(XA) = true \wedge agent(XB) = true\\ &\wedge number(XNA) = true\\ &\wedge member(sent(X, Y, encr(nonce(XNA), K)), XT) = false \Rightarrow\\ &m([sent(XA, XB, encr(nonce(XNA), key(XA, XB)))|XT]) = true.\end{aligned}$$

The symbol $encr(A, K)$ represents encryption of message A by key K. The $member$ predicate checks a *side condition*, i.e. that nobody has used the nonce XNA to start a run before. The unary predicates $number$ and $agent$ are for type checking. Agent names may be instantiated to $a, s(a), s(s(a)), \ldots$ and numbers to $0, s(0), \ldots$.

Principals should respond to $key(A, B)$ and $key(B, A)$, as they are in reality the same. At the moment we model this with two possible rules for message 2, but it should be straightforward to extend the model to give a cleaner treatment of symmetric keys as sets of agents:

$$\begin{aligned}&member(sent(X, XB, encr(nonce(XNA), key(XA, XB))), XT) = true\\ &\wedge m(XT) = true \Rightarrow\\ &m([sent(XB, XA, encr(s(nonce(XNA)), key(XA, XB)))|XT]) = true.\end{aligned}$$

$$\begin{aligned}&member(sent(X, XB, encr(nonce(XNA), key(XB, XA))), XT) = true\\ &\wedge m(XT) = true \Rightarrow\\ &m([sent(XB, XA, encr(s(nonce(XNA)), key(XB, XA)))|XT]) = true.\end{aligned}$$

Notice we allow a principal to respond many times to the same message, as Paulson's formalism does. The conjecture which gives rise to the attack is:

$$\begin{aligned}&member(sent(XA, XB, encr(nonce(XNA), K), XT) = true\\ &\wedge XT = [sent(X, XA, encr(s(nonce(XNA)), K)|T]\\ &\wedge member(sent(Y, XA, encr(s(nonce(XNA)), K), T) = false\\ &\wedge m(XT) = true \Rightarrow eq(X, XB) = true\end{aligned}$$

Informally this says 'For all valid traces T, if A starts a run with B using nonce N_A, and receives the reply $t(N_A)$ from principal X, and no other principal has sent a reply, then the reply must have come from agent B.' The proviso that only one reply has been received is to rule out a rather ineffective attack whereby the spy simply waits for B to reply and then copies B's response. The system is able to refute this conjecture, and the final line of output gives a binding for the variable XT:

```
c(sent(spy,a,encr(s(nonce(0)),key(a,s(a)))),
c(sent(a,s(a),encr(s(nonce(0)),key(a,s(a)))),
c(sent(spy,a,encr(nonce(0),key(a,s(a)))),
c(sent(a,s(a),encr(nonce(0),key(a,s(a)))),nil))))
```

This is exactly the attack suggested by Clark. Note that the attacks starts from the innermost position of the formula with message 1 of the standard protocol, and proceeds outwards. More details of our model for the problem, including the specification of intruder knowledge, can be found at `http://www.dai.ed.ac.uk/~grahams/linda`.

This application highlights a strength of our method, in that side conditions for messages to be sent or for the spy to construct fake messages rely on parts of the trace as yet uninstantiated at the time the rules are applied. Proofs propagate backwards from the conjectured property, as they do in Paulson's model, matching up and satisfying side conditions as they go. The side conditions influence the path taken through the search space, as smaller formulae are preferred by the default heuristic in the prover. This means that some traces a naïve counterexample search might find are not so attractive to our system, e.g. a trace which starts with several principals sending message 1 to other principals. This will not be pursued at first, as all the unsatisfied side conditions will make this formula larger than others.

It may also be possible to implement this formalism using conditional rewrite rules. However, for a backwards style proof, side conditions have to be decided on as yet uninstantiated traces. This may be problematical for some conditional rewrite systems. Otherwise, conditions must be carried forward, as they are in our system.

In future, we intend to implement more sophisticated heuristics to improve the search performance. These could include eager checks for unsatisfiable side conditions. Formulae containing these conditions could be discarded as redundant. Another idea is to vary the weight ascribed to variables and function symbols, so as to make the system inclined to check formulae with predominantly ground variables before trying ones with many uninstantiated variables.

Once we have a working implementation of the system in a faster prover like SPASS, we intend to run the system on larger protocols. A first goal is to rediscover the parallel session attack discovered by Paulson. The system should also be able to discover more standard attacks, and the Clark survey, (Clark and Jacob, 1997), provides a good set of examples for testing. We will then try the system on other protocols and look for some new attacks.

7 Further Work

Our first priority is to re-implement the system using SPASS, and then to carry out further experiments with larger false conjectures and more complex security protocols. This will allow us to evaluate the technique more thoroughly. Improvements in the new implementation will include the ability of a single consistency checker to check many clauses, as described in section 4, and inclusion of the heuristics mentioned in section 6.

The Comon-Nieuwenhuis technique has some remaining restrictions on applicability, in the particular the need for *reductive definitions*, a more relaxed notion of reducibility than is required for ground convergent rewrite systems. It is quite a natural requirement that recursive function definitions should be reducing in some sense. Even so, it should be possible to extend the technique for non-theorem detection in the case of non-reductive definitions, at the price of losing any reasonable chance of proving a theorem, but maintaining the search guidance given by the proof by consistency technique. This would involve allowing inferences by general superposition and equality resolution if conjecture superposition is not applicable.

Our main interest is to apply the technique to more complex security protocol examples. A key advantage of our security model is that it allows attacks involving arbitrary

numbers of participants. This should allow us to investigate the security of protocols involving many participants in a single run, e.g. conference key protocols.

8 Conclusions

In this paper we have presented a working implementation of a novel method for investigating an inductive conjecture with a view to proving it correct or refuting it as false. We are primarily concerned with the ability of the system to refute false conjectures, and have shown results from testing on a variety of examples. These have shown that our parallel inductive completion and consistency checking system requires considerably fewer clauses to be derived than a standard first-order prover does when tackling the whole problem at once. The application of the technique to producing attacks on faulty cryptographic security protocols looks promising, and the system has already synthesised an attack of a type many finite security models will not detect. We intend to produce a faster implementation using the SPASS theorem prover, and then to pursue this application further.

Acknowledgements

Thanks to Julia Degenhardt for translating some of Reif et al.'s technical report.

References

Bachmair, L. (1991). *Canonical Equational Proofs*. Birkhauser.

Bouhoula, A., Kounalis, E., and Rusinowitch, M. (1992). Automated mathematical induction. Rapport de Recherche 1663, INRIA.

Bouhoula, A. and Rusinowitch, M. (1995). Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14(2):189–235.

Burrows, M., Abadi, M., and Needham, R. (1990). A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36.

Carreiro, N. and Gelernter, D. (1989). Linda in context. *Communications of the ACM*, 32(4):444–458.

Clark, J. and Jacob, J. (1997). A survey of authentication protocol literature: Version 1.0. Available via `http://www.cs.york.ac.uk/jac/papers/drareview.ps.gz`.

Comon, H. and Nieuwenhuis, R. (2000). Induction = I-Axiomatization + First-Order Consistency. *Information and Computation*, 159(1-2):151–186.

Ganzinger, H. and Stuber, J. (1992). *Informatik — Festschrift zum 60. Geburtstag von Günter Hotz*, chapter Inductive theorem proving by consistency for first-order clauses, pages 441–462. Teubner Verlag.

H. Zhang, D. Kapur, M. S. K. (1988). A mechanizable induction principle for equational specifications. In Lusk, E. L. and Overbeek, R. A., editors, *Proceedings 9th International Conference on Automated Deduction, Argonne, Illinois, USA, May 23-26, 1988*, volume 310 of *Lecture Notes in Computer Science*, pages 162–181. Springer.

Huet, G. and Hullot, J. (1982). Proofs by induction in equational theories with constructors. *Journal of the Association for Computing Machinery*, 25(2).

Hutter, D. and Sengler, C. (1996). INKA: the next generation. In McRobbie, M. A. and Slaney, J. K., editors, *13th Conference on Automated Deduction*, pages 288–292. Springer-Verlag. Springer Lecture Notes in Artificial Intelligence No. 1104.

Jouannaud, J.-P. and Kounalis, E. (1989). Proof by induction in equational theories without constructors. *Information and Computation*, 82(1).

Knuth, D. and Bendix, P. (1970). Simple word problems in universal algebra. In Leech, J., editor, *Computational problems in abstract algebra*, pages 263–297. Pergamon Press.

Kounalis, E. and Rusinowitch, M. (1990). A mechanization of inductive reasoning. In Press, A. and Press, M., editors, *Proceedings of the American Association for Artificial Intelligence Conference, Boston*, pages 240–245.

Musser, D. (1980). On proving inductive properties of abstract data types. In *Proceedings 7th ACM Symp. on Principles of Programming Languages*, pages 154–162. ACM.

Paulson, L. (1998). The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6:85–128.

Protzen, M. (1992). Disproving conjectures. In Kapur, D., editor, *11th Conference on Automated Deduction*, pages 340–354, Saratoga Springs, NY, USA. Published as Springer Lecture Notes in Artificial Intelligence, No 607.

Reif, W., Schellhorn, G., and Thums, A. (2000). Fehlersuche in formalen Spezifikationen. Technical Report 2000-06, Fakultät fur Informatik, Universität Ulm, Germany. (In German).

Reif, W., Schellhorn, G., and Thums, A. (2001). Flaw detection in formal specifications. In *IJCAR'01*, pages 642–657.

Reif, W. (1995). The KIV Approach to Software Verification. In M. Broy and S. Jähnichen, editors, *KORSO: Methods, Languages and Tools for the Construction of Correct Software*, volume 1009. Springer Verlag.

Weidenbach, C. (1999). Towards an automatic analysis of security protocols in first-order logic. In Ganzinger, H., editor, *Automated Deduction – CADE-16, 16th International Conference on Automated Deduction*, LNAI 1632, pages 314–328, Trento, Italy. Springer-Verlag.

AISB Journal **1**(2), © SSAISB, 2002
http://www.aisb.org.uk

Stochastic Simulation of Inherited Kinship-Driven Altruism

Heather Turner and Dimitar Kazakov

Department of Computer Science, University of York
Heslington, York, UK
hrt103@cs.york.ac.uk ; kazakov@cs.york.ac.uk

Abstract

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 in the context of a multi-agent system simulating natural selection. Classical Darwinism and Neo-Darwinism are compared, and the principles of the latter are implemented in the system. The experiments study the factors that influence the successful propagation of altruistic behaviour in the population. The results show that the natural phenomenon of kinship-driven altruism has been successfully replicated in a multi-agent system, which implements a model of natural selection different from the one commonly used in genetic algorithms and multi-agent systems, and closer to nature.

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

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 maximum population size
Preservation of energy	enforced	not considered

(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 us to avoid the use of an explicit fitness function, and instead 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 forcing 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. 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 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).

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 its 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:

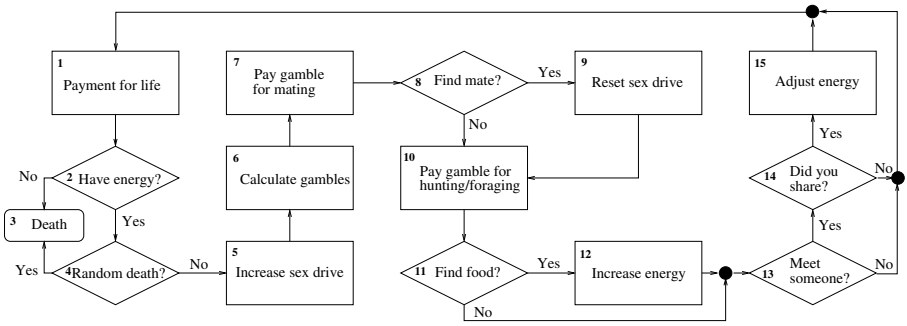


Figure 1: Simulation outline

1. Make a payment of energy to the environment (energy expended to survive generation).
2. If all energy is used up, the individual dies.
3. Individual has 'died' and is therefore removed from the population.
4. Random death occurs with some probability for each individual (this probability increases exponentially with age).
5. Increase sex drive, and 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, if any of the available energy to expend in search of a mate or food.
7. The gamble for mating is 'paid' to the system.
8. Pairs selected at random from the mating pool are deemed to have 'met' with some probability. Each must satisfy certain energy requirements, and the pair must not be related as parent and child. The probability that they mate is set in proportion to their mating gambles and determines whether or not they actually produce offspring. 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 (or *food* gamble) 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.

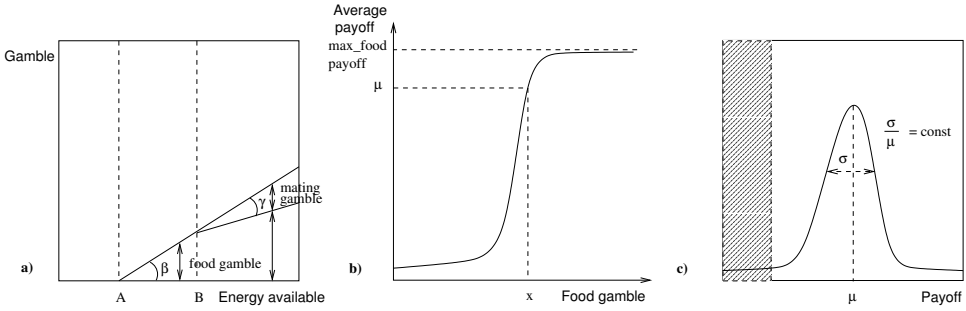


Figure 2: **(a)** Computing the food and mating gambles from the available energy. **(b)** Mapping food gambles to average food obtained. **(c)** Distribution of the amount of food received for a given average payoff μ.

Gambling policies The searches for a mate and food are modelled as stochastic processes, in which an individual spends (or "gambles") a certain amount of its energy, and receives a payoff from the environment (finds a mate or a given amount of food) with a certain probability. The functions described in Table 2 and displayed in Figure 2a are used to compute the food and mating gambles. The mating gamble is used as described above. The actual amount of food received from the environment is determined from the food gamble in the following way. Firstly, the sigmoid function from Equation 1 is used to compute the average number of units of food/energy μ that the effort represented by the food gamble will produce (see Figure 2b).

$$\mu = \frac{max\_food\_payoff}{1 + e^{-0.025*(gamble\text{-}200)}} \tag{1}$$

The actual amount of food is then generated at random according to a Gaussian distribution $G(\mu, \sigma)$ (Figure 2c) where the ratio σ/μ is kept constant for all μ to ensure that

Table 2: Computing gambles from the energy available

if Energy $\leq$ A
Food Gamble := 0
Mating Gamble := 0
if A $<$ Energy $\leq$ B
Food Gamble := $tg\,\beta$ * (Energy - A)
Mating Gamble := 0
if Energy $>$ B
Food Gamble := $tg\,\beta$* (B - A) + $tg\,(\beta - \gamma)$ * (Energy - B) =
$tg\,\beta$ * (Energy - A) - $tg\,\gamma$ * (Energy - B)
Mating Gamble := $[tg\,\beta - tg(\beta - \gamma)]$ * (Energy - B) = $tg\,\gamma$ * (Energy - B)

only a very small, fixed proportion of the payoffs are negative; these, when generated, were reset to zero.

The parameters of the gambling function, that is, A, B, $tg\,\beta$ and $tg\,\gamma$, are encoded in the genes of the individuals, and, therefore, are subject to natural selection.

The above discussion shows that in this simulation spatial phenomena (food discovery, encounter with another individual) are represented as random processes with a certain probability. It is worth noting that physical distance between individuals is ignored, and the encounter of each pair is equally probable. Similarly, the probability of finding food does not depend on the past actions of the agent, as it would be the case if its co-ordinates were taken into account.

4 Experiments and Evaluation

The tool specified in the previous section 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 studies 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*).

The Royalty kinship recognition policy assumes one knows its relatives and the degree to which they are related. Each individual keeps a record of their relatives up to two levels up and down the genealogical tree (see Figure 3). Instead of recording the actual relationship, relatives are grouped in two sets, according to whether on average they share 50% or 25% of their genes with the individual in question. The first group consists of parents and children, the second of grandparents, grandchildren, and siblings. Treating siblings in this way can be explained by the fact that individuals change partners in every generation, and, therefore, the vast majority of siblings are actually half-sibs, which is the case displayed in Figure 3. One peculiarity of our implementation is that when two individuals mate, they produce exactly two children, the chromosomes of which are produced from the parents' by crossover. This means that if one child inherits a copy of a gene

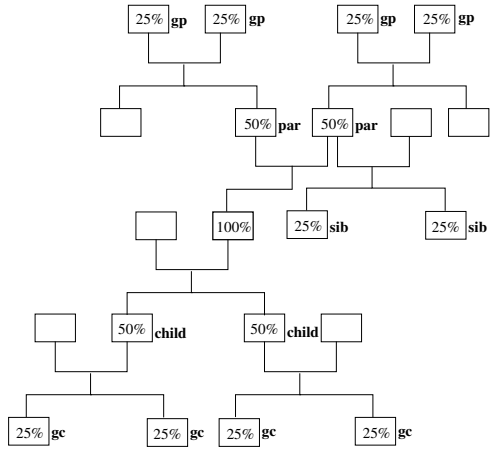


Figure 3: Average expected percentage of shared genes between relatives.

from one parent, the other child will *not* have that gene, unless the other parent carried it. In any case, the likelihood of two individuals mating together on more than one occasion is negligible in larger populations and the case of full-sibs is therefore discounted for simplicity in this implementation. Individuals who do not appear in either of the above groups of relatives are treated as being no relation at all.

The Prediction kinship recognition policy assumes that all genes but one (coincidentally, the one identifying altruistic individuals) are visible in the phenotype. A simple linear metric is then used to measure the similarity between the visible parts of genotype of the two individuals.

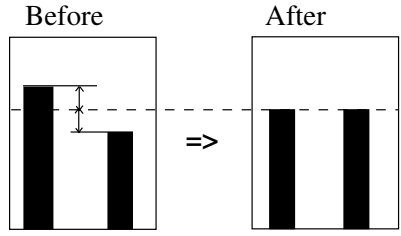


Figure 4: Sharing between identical twins: Communism

Type of sharing function Three social models are considered. *Communism* equalises the energy levels of two individuals with the same genome (see Figure 4). *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 correspond 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 each of the nine possible experiments three times (see Table 3). All parameters of the sharing func-

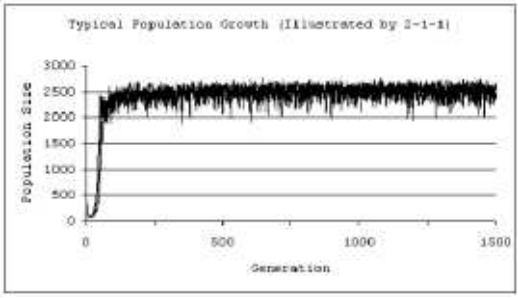
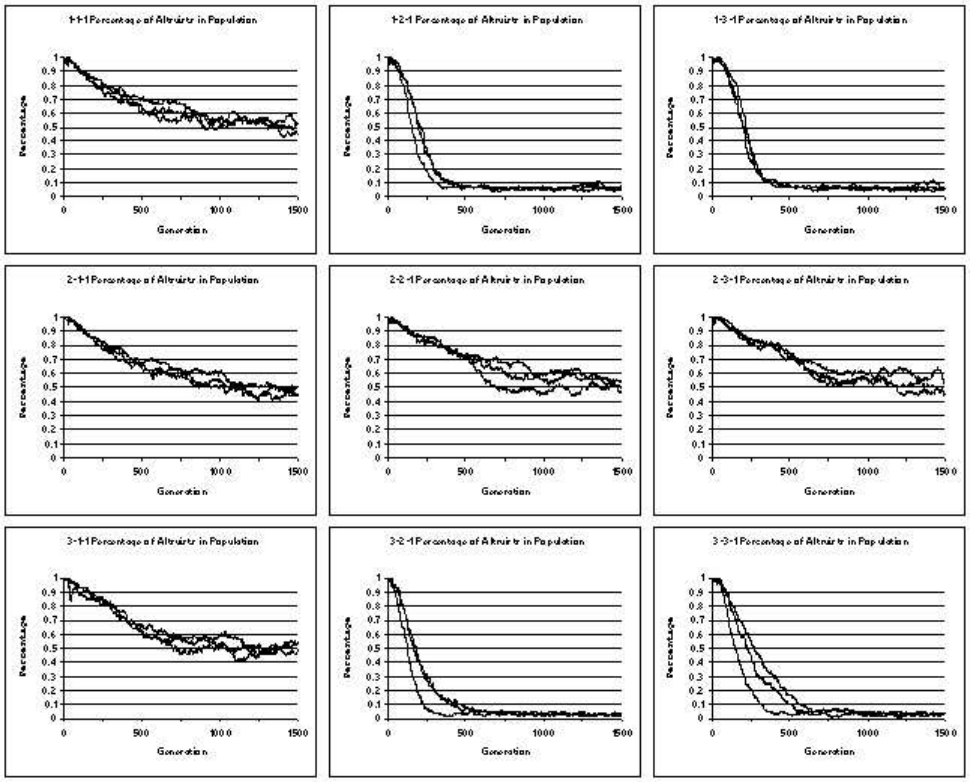


Figure 5: Evolution of population size

Table 3: 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.)



tions (α, θ), resp. pt were initially set at random, and left to evolve. When employing the Unknown model of kinship, a rather optimistic assumption was made, under which the donor treated the aid receiver as a parent or child.

The graphs in Table 3 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. 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 5 on the case of Royalty with Progressive Taxation.

Degree of altruism and availability of resources 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 range, is considered a more appropriate measure. The idea in the case of progressive taxation is shown in Figure 6 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.

Changes in the level of resources available in the system will by definition have an effect on the carrying capacity (maximum population size) of the environment, and could be expected to cause variations in the system dynamics, and possibly the ability of the environment to support altruism. We ran several experiments to see how the degree of altruism in the system varies for the different sharing policies (note that this level does not change, and is considered equal to 100% for the Communist sharing policy, so the graphs are omitted) with different amounts of energy (resources) available. The graphs in Table 4 indicate that altruism tends to converge faster to a single stable level when more energy is provided by the environment.

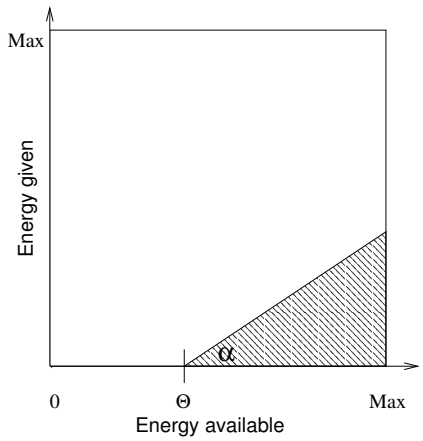
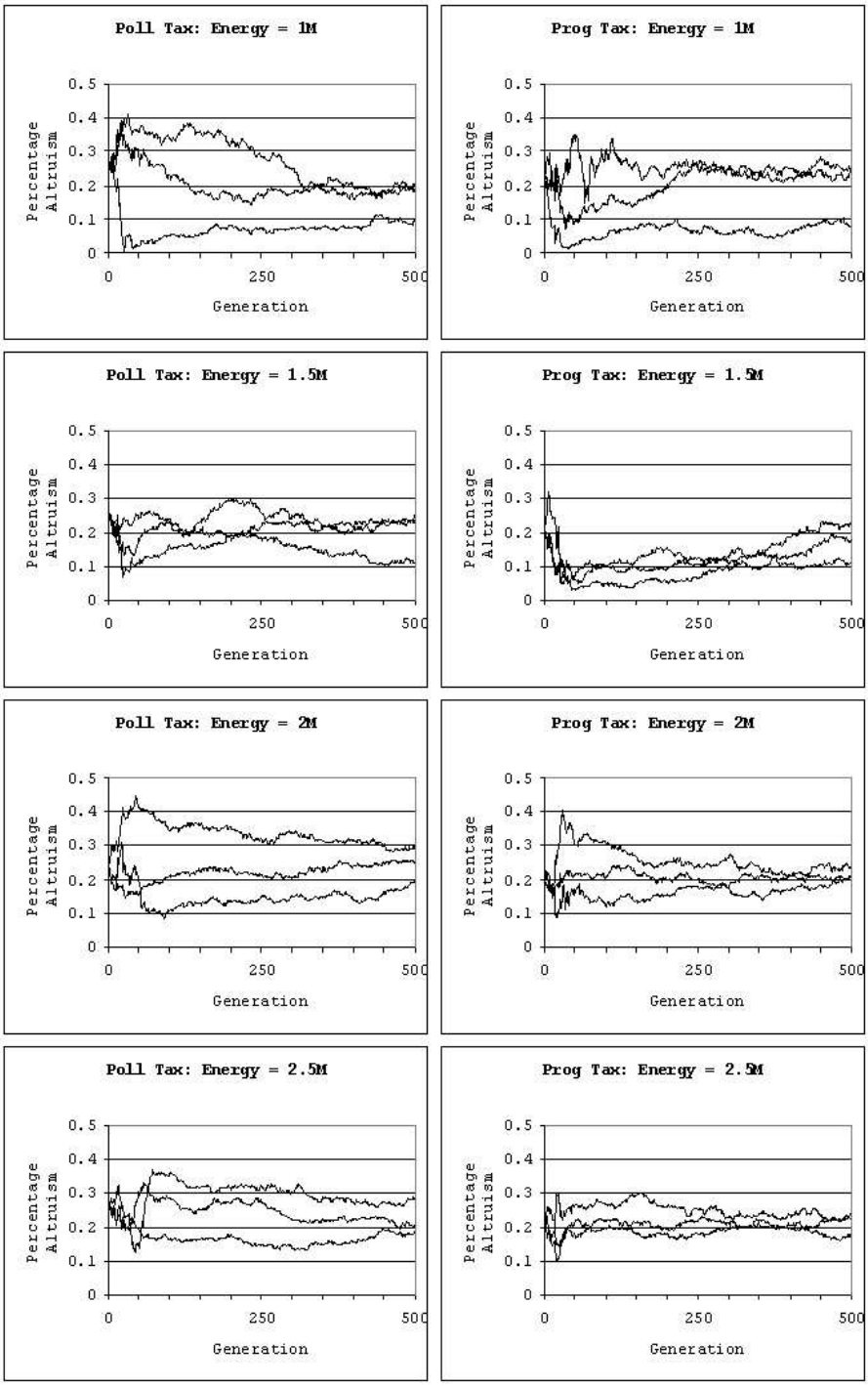


Figure 6: Measure of altruism

Table 4: Percentage of altruism (1=100%) evolving in the population as the sharing strategy and level of energy (resources) available are varied.



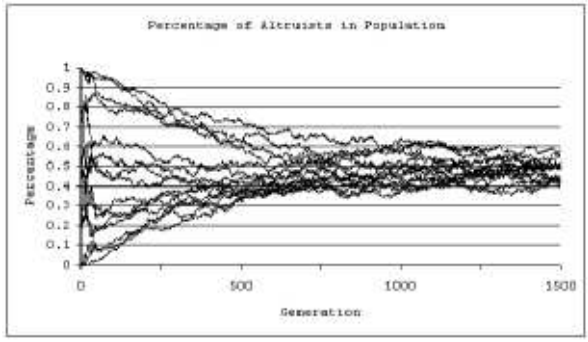


Figure 7: Percentage of altruists in the population with respect to initial levels (1=100%)

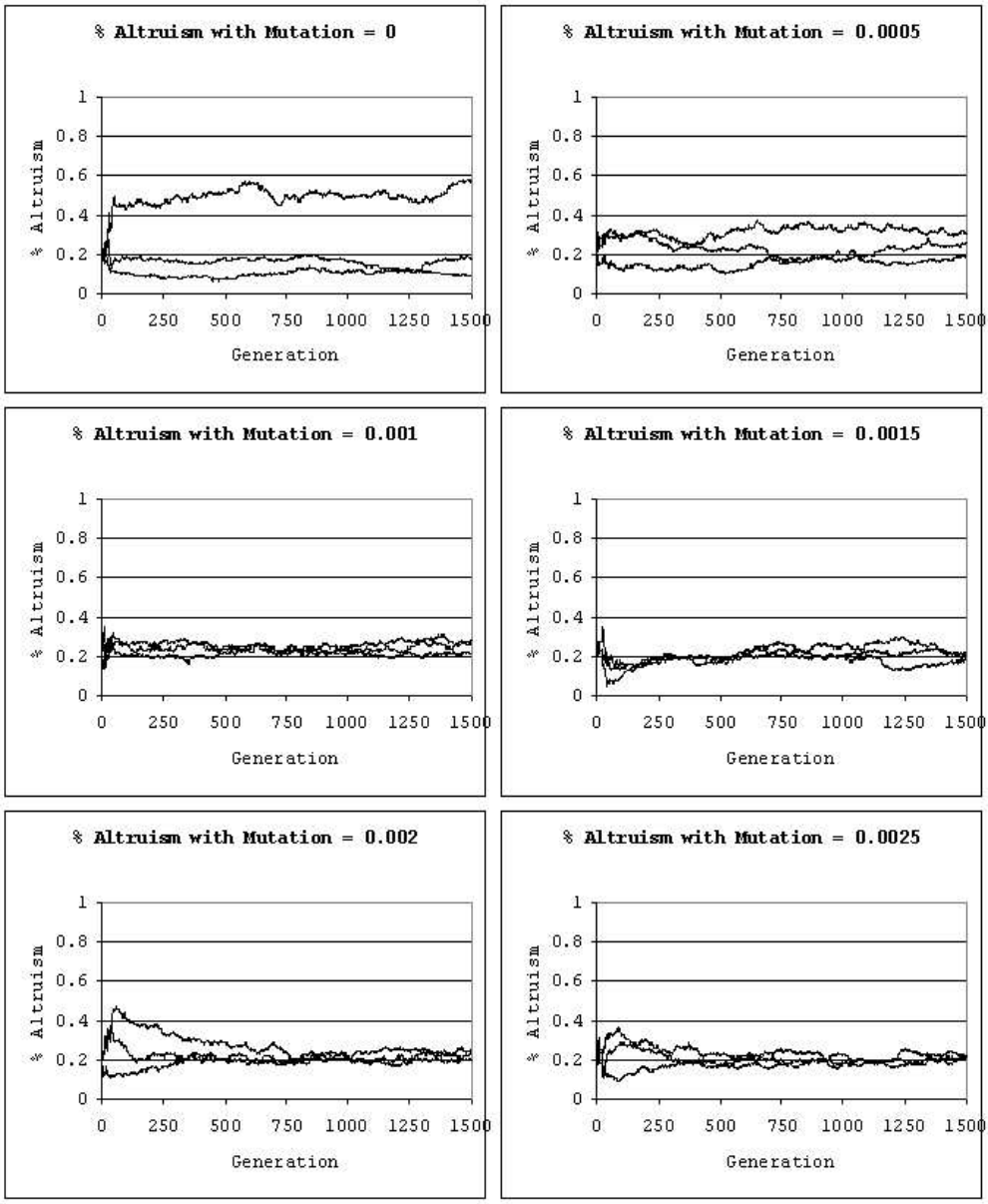
Initial ratio between altruistic and selfish individuals To study the influence that the initial proportion of altruistic to selfish individuals has on the levels of altruism selected by evolution, the Royalty with Progressive Taxation experiment was run with several initial values for this ratio. The results in Figure 7 show that the system reaches a dynamic equilibrium which, in the cases shown, does not depend on the initial ratio.

Mutation We conducted some experiments where the rate of mutation in the system was varied. Although it was maintained at relatively low levels, variation was seen in the speed of convergence to a stable level of altruism and the eventual level. The mutation rates were set at: 0; 0.0005; 0.001; 0.0015; 0.002 and 0.0025, with other variables fixed as follows: sharing function = Progressive Taxation, kinship-recognition policy = Royalty and Energy = 2.5M (see Table 5). At the lowest rates of mutation, there appears to be a greater variation in the evolved levels of altruism between runs of the experiment, making it difficult to draw conclusions about the rate of convergence. As the mutation rate increases, a more definite level of altruism is evolved, and the experimental populations converge faster to this level. It is unlikely that this trend would continue as the mutation rate increases much higher, since, at some point, the high level of mutation is likely to override the effects of natural selection. (For the third chart, where the level of mutation is at 0.001, note that it is just an extension of chart eight in Table 4: Progressive Taxation with 2.5M energy units, the same experimental setup, but run for three times as long.)

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 Barton's work could be made as his detailed results were not available in a suitable form. However, a few major 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 virtually the same size. Barton treats altruists and non-altruists as two different species, which in turn results in one species completely taking over the other

Table 5: Effect of varying mutation rate on the percentage of altruism in the population (1=100%).



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 that 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 up to experts in the relevant fields to judge.

The two 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 left 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 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: rich meet (mostly) rich, poor meet poor, individuals with high sexual drive are grouped together, etc.

In addition to the impact of resource availability and rates of mutation, studied in this paper, another environmental parameter, 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), would 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

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

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

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

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

Kazakov, D. and Kudenko, D. (2001). *Multi-Agent Systems and Applications*, chapter Machine Learning and Inductive Logic Programming for Multi-Agent Systems, pages 246–270. LNAI 2086. Springer.

Schröder, M. and Noy, P. (2001). Multi-agent visualisation based on multivariate data. In *Working Notes of the Fourth UK Workshop on Multi-Agent Systems UKMAS-01*.

Thompson, D., editor (1996). *The Oxford Compact English Dictionary*. Oxford University Press.

Turner, H. (2002). Stochastic simulation of inherited kinship driven altruism. Project report for a degree in Computer Science, University of York. Project supervisor: Dimitar Kazakov.

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

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

AISB Journal **1**(2), © SSAISB, 2002
http://www.aisb.org.uk

Using agents to build a distributed calculus framework

Philippe Mathieu, Jean-Christophe Routier, Yann Secq

Equipe Systèmes Multi-Agents et Coopération
Laboratoire d'Informatique Fondamentale de Lille
CNRS UPRESA 8022
Université des Sciences et Technologies de Lille
{mathieu, routier, secq}@lifl.fr

Abstract

This article argues that multi-agent systems can be seen as an interesting paradigm for the design and implementation of large scale distributed applications. Agents, interactions and organisations, that are the core of multi-agent systems, consitute a well fitted support to the design of complex real-world software. They combine the well defined and desired properties for good software engineering: decomposition, abstraction and organisation.

This paper presents an agent framework RAGE, *Reckoner AGEnts*, for an *easier* design of distributed calculus applications. This framework has been developped using the MAGIQUE multi-agent framework. Because distribution, cooperation and organization are key concepts in multi-agent systems, they are natural solutions for the design of such applications. The use of MAGIQUE has contributed to an easier development of RAGE and provides to the application the capacity to smoothly evolve and adapt.

In the first part of this paper, we introduce concepts that are emphasized in the multi-agent domain. Then, we present the principles of the MAGIQUE framework. The second part describes RAGE framework and details a practical example. Finally, the last part illustrates how an agent oriented approach has been followed for the design and implementation of RAGE.

1 Introduction

One application field of multi-agent systems (MAS) is *Distributed Problem Solving*, and one typical example of such problems is the distributed calculus. Because distribution, cooperation and organizations are key concepts in MAS, they are natural solutions for the design of applications of distributed computing.

The use of a multi-agent framework allows the designer to get rid of the problems that would have been generated by the distribution and the communications between (what would have been) clients. Indeed these are primitives in multi-agent infrastructures and are often even hidden to the designer. Moreover, the agent approach (or paradigm) leads the designer to a natural decomposition of the problem in term of *agents*, *tasks* and *roles*. We will not give here another definition of "what is an agent"[1], but we will stick to the

[1] For a lot of definitions of "agent", have a look on (Franklin and Grasser, 1996)

one proposed in (Wooldridge, 1997): *an agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives*. The notion of *role* has been largely emphathized in works related to actor languages (Hewitt, 1979), subject oriented programming (Harrison, 1993), and of course in the multi-agent field (Ferber and Gutknecht, 1999). This notion relies on the specification of the behaviour of an actor/agent. In a sense, roles can be seen as an equivalent of interfaces (or pure abstract classes) in object oriented programming. The main difference resides in the fact that interactions are not constrained with interfaces (anybody can invoke a method), while with *roles* one can ensure that the request comes from the right *role*. The other point with *roles* is that they identify the functional requirements and are not tied to particular agents.

For all these reasons we claim that multi-agent systems are appropriate infrastructures for the design and development of flexible frameworks for distributed calculus applications.

Clustering computers to share their power and exploit their idleness is an idea that has gain momentum with the advent of Internet (Gray and Sunderam, 1997; Stankovic and Zhang, 1998). Projects like distributed.net[2] or SETI@Home (Anderson and al., 1999) illustrate this trend. Despite their interest, they are centralized (they rely on a client-server approach) and they are monolithic based frameworks (clients are mono-calculus). RAGE instead proposes a framework where clients does not only provide computing power but can also create their own calculus. This approach is closer to frameworks like xDu(Gregory, 1998) or JavaParty (Philippsen and Zenger, 1997).

In the first part of this paper we present, briefly, the MAGIQUE framework. It is based on an organisational model and on an minimal agent model which have been put into concrete form as a JAVA API. MAGIQUE has been used to build the *easy* distributed computing environment application described in the second part: RAGE. We will describe both how this application has been designed and the resulting framework.

2 MAGIQUE : a Multi-Agent framework

MAGIQUE proposes both an agent model(Routier et al., 2001), which is based on an incremental building of agents, and an organizational model (Bensaid and Mathieu, 1997), based on a default hierarchical organization.

2.1 The agent model: building agents by making them skilled.

The agent model is based on an incremental building of agents from an elementary (or atomic) agent through dynamical skill acquisition. A skill is a *coherent set of abilities*. We use this term rather than service[3], but you can consider both as synonyms here. From a developer point view, a skill can be seen as a software component that gathers a coherent set of functionalities. The skills can then be built independently from any agent and reused in different contexts.

We assert that only two prerequisite skills are necessary and sufficient to the *atomic agent* to evolve and reach any wished agent: one to interact and one to acquire new skills(Routier et al., 2001).

[2] `http://www.distributed.net/`

[3] We keep *service* for "the result of the exploitation of a skill".

Thus we can consider that all agents are at birth (or creation) similar (from a skill point of view): an empty shell with only the two above previously mentioned skills.

Therefore differences between agents are issued from their *education*, i.e. the skills they have acquired during their *existence*. These skills can either have been given during agent creation by the developer, or have been dynamically learned through interactions with other agents (now if we consider the programmer as an agent, the first case is included in the second one). This approach does not introduce any limitations to the abilities of an agent. Teaching skills to an agent is giving him the possibility to play a particular role into the MAS he belongs to.

This paradigm of dynamic construction of agent from skills, has several advantages :

- *development becomes easier* : modularity is given by the skills,
- *efficiency* : skills distribution can be dynamically adapted,
- *robustness* : critical skill can be preserved,
- *autonomy and evolutivity* : runtime customization available through adaptation to runtime environment.

2.2 The organisational model.

In MAGIQUE, there exists a basic default organisational structure which is a hierarchy. It offers the opportunity to have a default automatic delegation mechanism to find a skill provider.

The hierarchy characterizes the basic structure of acquaintances in the MAS and provides a default support for the routing of messages between agents. A hierarchical link denotes a communication channel between the implied agents. When two agents of a same structure are exchanging a message, by default it goes through the tree structure.

With only hierarchical communication, the organisation would be too rigid, thus MAGIQUE offers the possibility to create direct links (i.e. outside the hierarchy structure) between agents. We call them *acquaintance links* (by opposition of the default *hierarchical links*). The decision to create such links depends on some agent policy. However the intended goal is the following: if some request for a skill occurs frequently between two agents, the agent can take the decision to dynamically create an acquaintance link for that skill. The aim is of course to promote the "natural" interactions between agents at the expense of the hierarchical ones.

With the default acquaintance structure, an automatic mechanism for the delegation of request between agents is provided. When an agent wants to exploit some skill it does not matter if he knows it or not. In both cases the way he invokes the skills is the same. If the realization of a skill must be delegate to another, this is done transparently for him, even if he does not have a peculiar acquaintance for it. The principle of the skill provider search is the following:

- the agent knows the skill, he uses it directly
- if he does not, several cases can happen
  - if he has a particular acquaintance for this skill, this acquaintance is used to achieve the skill (ie. to provide service) for him,
  - else he is a supervisor and someone in his sub-hierarchy knows the skill, then he forwards (recursively through the sub-hierarchy) the realisation to the skilled agent,

- else he asks its supervisor to find for him some skilled agent and his supervisor applies the same delegation scheme.

One first advantage of this mechanism of skill achievement delegation is to increase the reliability of the multi-agent system: the particular agent who will perform the skill has no importance for the "caller", therefore he can change between two invocations of the same skill (because the first had disappeared of the MAS or is overloaded, or ...).

Another advantage appears while developping applications. Since the search of a skilled agent is automatically achieved by the hierarchy, when a request for a skill is programmed, there is no need to specify a particular agent. Consequently the same agent can be used in different contexts (i.e. different multi-agent applications) so long as an able agent (no matter which particular one) is present. A consequence is, that when designing a multi-agent system, the important point is not necessarily the agents themselves but their skills (ie. their roles).

2.3 The API

These models have been put into concrete form as a JAVA API, called MAGIQUE too. It allows to develop multi-agent systems distributed over heterogeneous network. Agents are developed from incremental (and dynamical if needed) skill plugging and multi-agent system are hierarchically organized. As described above, some tools to promote dynamicity in the MAS are provided: direct acquaintance links can be created, new skills can be learned or exchanged between agents (with no prior hypothesis about where the bytecode is located, when needed it is transferred between agents). The API, a tutorial and samples applications can be downloaded at `http://www.lifl.fr/MAGIQUE`.

3 RAGE: an easy distributed computing framework

In this section, we will illustrate how we have designed and implemented RAGE using the MAGIQUE infrastructure. We will see that an agent oriented approach offers simplicity in the development of such an application, and promotes an easy evolution of the system too.

We will briefly present the framework, that is the point of view of the framework user, then we will see how it has been designed, which is the point of view of the designer. While the end user can see only distributed entities, he can name them agents or distributed objects if he prefers, the designer clearly tackles the problem using agent notions: agents, interactions and organization.

3.1 The *Rage* framework: the end user point of view

RAGE[4] wants to be a framework that offers an easy development of distributed calculus, that allows multi-applications computing in parallel and that can scale with the available power (cluster).

Simplicity was a preliminary condition for the framework. The main idea was to provide an easy framework for non computer scientists. To achieve this goal, we had to define a small number of concepts that the user has to understand in order to feed the system with its calculus. It is important to note that the end user has no need to know

[4] RAGE stands for Reckoner AGEnt

anything about agent or multi-agent systems, even if he has to understand at least some object-oriented notions (since he must inherits some predefined patterns). The user has mainly to know two concepts: what is a *task* and what is a a *result*. A *task* can be seen as the algorithm that is distributed, while a *result* represents the data produced by the *task* and which must be stored.

The user of the framework has to define what should be distributed, and what is the global scheduling of its main algorithm. Let us take the example of a matrix that has to be computed and then used to solve a linear system. The main algorithm could be cut in two phases: first compute the matrix (distributed), then solve the system (centralized). In the first stage the user dispatches the tasks that populate the matrix, then he retrieves the matrix to solve the system. The *task* defines the chunk of algorithm that will be distributed among agents. The simplest way for the user to create a *task* is to subclass the `AbstractTask` class and to define the only two methods:

```
abstract public void compute();
abstract public boolean finished();
```

The complexity for the user is then not bigger than writing a JAVA Applet. Therefore, the end user has a rather OO view of the framework: he extends one class to tailor it to his distributed application and uses the underlying framework without necessary explicitly knowing what happens.

To ease the migration or cancellation of running tasks, the user has to follow a simple principle while designing the `compute()` method : tasks are considered as cooperative in respect to the threading model[5]. The figure 1 illustrates this principle and describe the lifecycle of a `Task`.

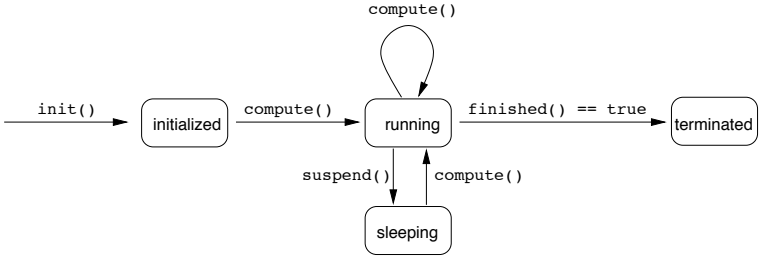


Figure 1: The lifecycle of a RAGE *Task*.

To illustrate what the user must do, we will detail a simple example : computing an approximation of the π number. The idea of the algorithm is to randomly choose points in an unitary square, and to check wether the point is within the quarter of circle or not. Then, we can apply the following formulae : $\pi = 4 * I/N$, where N is the total number of points and I is the number of points within the quarter of circle (see figure 2).

The accuracy of this algorithm increases when N grows. Thus, the user will have to define a `PiTask` that encapsulates this algorithm, to create a set of these tasks and to dispatch them within RAGE. The complete source code for the `PiTask` class is given here.

[5]This model was inspired by works done on the FAIRTHREADS project at `http://www-sop.inria.fr/mimosa/rp/FairThreads/`

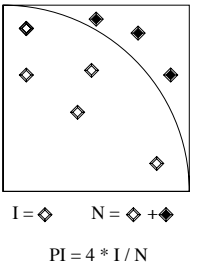


Figure 2: A Monte-Carlo algorithm to compute an approximation of π

```
public class PiTask extends AbstractTask {
  public Double pi;
  private int crtIter = 0, inner = 0, niter, chunk;
  public PiTask(String factId, String taskId, int niter){
    super(factId, taskId);
    this.niter = niter; chunk = niter/10;
  }
  public void compute(){
    double x, y;
    for(; (crtIter % chunk) < chunk; crtIter++){
      x = Math.random(); y = Math.random();
      if (Math.sqrt(x*x + y*y) <= 1.0)
        inner ++;
    }
    pi = new Double(4*((double)inner/(double)niter));
  }
  public boolean finished(){
    return crtIter == niter;
  }
  public double percent(){ // Percent of progress
    return (double) (crtIter * 100)/(double)niter;
  }
}
```

Despite its simplicity, this example illustrates the programming model of the RAGE framework :

- extend `AbstractTask` (or implement the `Task` interface),
- implement the `compute()` and `finished()` methods,
- tag as `public` members that should be saved as result.

Then, the user creates a bunch of tasks and sends them to the framework. Later, he can retrieve the results and compute the average value. One interesting point of the `compute` method is that it illustrates the delegation of control from the task to the agent that handles it. If the agent receives an order to terminate or migrate the task, it can be handled only if the user releases the control to the agent. Thus, a `PiTask` will need 10 calls to the `compute` method before being achieved. The granularity of the `compute` method should therefore be carefully studied by the user if he wants to take advantages of task migration or termination.

3.2 Implementation with MAGIQUE: the designer point of view

The design of the framework has loosely followed the GAIA methodology (Wooldridge et al., 2000) and has been defined through those steps:

- firstly, the definition of the roles involved in the framework and their abilities (or skills)
- secondly, the definition of the organization of roles within the system
- and finally, the mapping of roles on agents.

The first step identifies the main entities of the application, and the abilities they have to assume. This step describes also the interactions between roles. The second step defines the number of agents playing a particular role and their position in the organization. Then, the last step do the mapping between the agents and the role(s) they play.

The first task is to define the roles. A short analysis of the problem allows to determine a set of roles that can be distinguished to bring a solution to the problem of distributed calculus. Here is a short description of these roles:

Boss role is responsible of all the interactions with the user part

Task Dispatcher role has to dispatch *tasks* and deal with fault tolerancy.

Platform Manager role manages the agents that will compute the *tasks*, and must ensure that there are always available *tasks* for these agents.

Reckoner Agent role is the worker of the framework, it computes *tasks*, and send back the *results* of this computation.

Repositories Manager role manages the storage of *results*.

Result Repository role is a mirror of the database of *results*.

As we implemented the system with MAGIQUE, we have put in concrete form those roles with the corresponding skills. For example, the *Platform Manager role* is defined by a MAGIQUE skill that implements its goal: getting tasks and dispatching them to *Reckoner Agents*. A role is then defined by a set of skills that forms its functionnalities. The interaction between the roles must then be described. With MAGIQUE, the dynamic of interactions between roles is contained in the skills.

Now that we have seen the involved entities, let us have a look on their interactions, that is in the backstage of the application. While the user "runs" its calculus, he first sends his *tasks* to the framework, then they are stored until an agent requests them. Once the agent has computed its *task*, he sends the *result* of the computation back to a repository. Thus the user can retrieve them and go on with its main algorithm. For the user, distribution and computation are totally abstract, he only has to feed the framework with *tasks* and retrieves the *results* : agents of the framework manage everything for him.

This stage can be considered as the analysis phase. We have to define the abilities associated to a role. This leads to the definition of a skill interface. It is important to work on interfaces, as skill implementations are not considered at this stage. Actually, the implementation will be dependant of the available runtime : we will indeed not have the same skill running on a workstation and on a cellular phone, but the interface of the skill will be the same.

Once those entities are identified, the architecture of the system must be defined. When working with MAGIQUE, it means that the architecture is shaped by the logical

grouping of roles and by the dynamic of interactions. If we take a look back at interactions: users send *tasks* to the *Boss*, which sends them to the *TaskDispatcher*. Then, *PlatformManagers* that are available request *tasks* and dispatch them to available *ReckonerAgents*. Once the computation of a *task* is done, the *result* is sent to the *Repositories Manager*. The figure 3 shows the logical hierarchy of roles, it does not define how roles

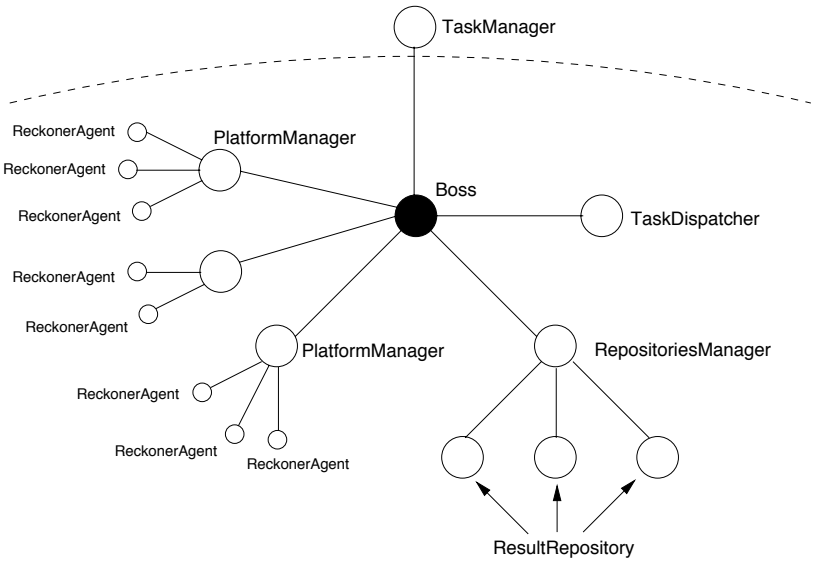


Figure 3: Rage's organization: the hierarchy of roles.

are mapped onto agents and how agents are mapped onto the network of computer, but instead provides the topology of what we call *natural acquaintances* structure : the default organization.

There is no a priori reason to respect a "one role–one agent" rule, the whole hierarchy could even be mapped onto one single agent (that would not be useful, but it could be done!). It is important to note that we could have applied the same role decomposition with another agent infrastructure, and then used another topology of organisation. If we had used Madkit (Gutknecht O., 2000) for example, the organisation would have followed the Aalaadin model (Ferber and Gutknecht, 1998), and thus we whould have had organize roles with groups instead of the hierarchy.

Here we have briefly seen the designer vision: the determination of roles and their interactions and then the choice of the organization. The agent oriented approach has allowed a quick development of the RAGE framework. Moreover, we see that even if finally the user has not necessarily an agent vision of the framework, the design was agent oriented. And while the framework is running, agents are working on behalf of RAGE users.

3.3 Experiments done with RAGE

To evaluate the framework, we have done some experiments that are ranging from simple examples to complex applications. The first experiment is a *tutorial* example that we have

seen in section 3.1 (i.e. computation of an approximation of π). It is based on a Monte-Carlo method and illustrates how simple it is to create a *task* and to run the framework.

The second tutorial experiment is a *naive* implementation for prime number decomposition, which enabled us to evaluate the scaling of the infrastructure. The idea is to test the primality of a number by making an exhaustive search of its factors.

The third example is bigger : it is an exploration of the underlying structure of the Donkey sliding block game. The algorithm consists in an exhaustive generation of game states graphs(Elwyn R. Berlekamp, 1982). It is interesting as it works in two stages and shows how computations can be canceled. The first stage consists in the creation of all valid states of the game. Then, we choose randomly a state that will be considered as a seed to create the graph of all accessible states from it. This is where the cancellation of computation is needed : several seeds are chosen and computations take place, but sometimes we have to check that we are not computing the same graph from different seeds. Thus, when a graph G is complete, each task receives the set of states of G and can therefore check if they are computing the same graph or not.

The last example is an application for solid mechanics : it is an implementation of the two-dimensional displacement discontinuity method (Crouch and Starfield, 1983). This sample provides a way to introduce the idea of shared data because each task needed to access a matrix representing forces. This mecanism of shared data is handled by *PlatformManager*s : they retrieve data from the user and make them available to their *ReckonerAgent*s.

These experiments along with the framework can be downloaded at `http://www.lifl.fr/SMAC/projects/magique/examples`.

3.4 What is next on the framework ?

The RAGE framework misses some features that could ease its use and broaden the field of applications it can handle. In this section, we present the most needed enhancements, firstly within the framework himself, and secondly around it.

The main missing feature in RAGE is the lack of inter-tasks communication. It has not yet been implemented, but thanks to the underlying multi-agent system, it can be quite easily handled. As tasks are managed by *ReckonerAgent*s, inter-tasks communication could be provided by using communication paths provided by the hierarchy. It would involve the implementation of a search algorithm between *PlatformManager*s : if task identification is done through their identifier, the implementation is straightforward, but not really usable. It would be better that each task provides some meta-description, and that the search algorithm works on these description to do the matching.

Another improvement that is not yet implemented, but could easily be added is multi-application scheduling. Several computations can be done simultaneously within RAGE, and now the *TaskDispatcher* distributes tasks without any consideration on them. We could add some scheduling policies, thus it could be possible to distribute twice much tasks from one application than another for example. Or, we could even introduce some kind of economical policy(Wolski et al., 2000) where users that provide more computation power to the framework by running *PlatformManager*s, could have a greater priority.

A last important feature that should be implemented concerns the database of results. At present, we are using an object database to ease the deployment of the framework, and there is only one database of results, which is managed by a *ResultRepository* agent. As results can be queried by users and also by tasks that are being computed, we plan to implement some automated mirroring of the database of results. The hardest part will be to establish metrics that could provide some criteria to the framework about the usage of

the database. But if we have these criteria, mirroring is easily implemented by dynamical skill acquisition : a new agent is created where needed, the *ResultRepository* role is given to him and then we need to synchronize the local database with others. Another approach could be to implement an automatic mirroring of the database by using some kind of replication mecanisms based on peer-to-peer interactions like the Freenet system(Clarke et al., 2000). The main advantage of this approach is that no metrics have to be defined, replication is only dependent on the usage of data.

Finally, we plan other enhancements around the framework : mainly on tools that can be provided and on packaging. At present, we have mainly worked on the framework, and thus we do not provide any tools to help users to visualize the dynamic of RAGE. An administration tool, that could provide feedback to the user and the ability to dynamically manage tasks and results, would be really useful. The packaging of the framework should be enhanced : we could provide one bundle for the framework, another for computational clients (basically, a *PlatformManager* and its sub-hierarchy). And, it could be interesting to deliver this client bundle with the JavaWebStart[6] technology, so new clients could easily join a running framework through their browsers.

4 Conclusion

The presented application, RAGE, is an easy to use framework for distributed computing intended to non-compter scientists. It allows to develop, distribute and run calculus over an heterogeneous framework with no need of background in distributed computing or agent oriented technologies. The user can only direct his efforts toward his very calculus.

This article argues that multi-agent paradigms: agent animacy, organization and roles, are key notions to support the analysis, design and implementation of open large-scale distributed systems. It is particulary significant that these application models are in adequation with multi-agent paradigms. It has been illustrated with an application of distributed calculus, but could be extended to other classes of applications like Computer Supported Collaborative Work(Routier and Mathieu, 2002)).

From our point of view, RAGE demonstrates that agent oriented programming is an appropriate framework for the development of such distributed applications. And as a consequence, the resulting framework is easy, first, for the user who has a calculus to do, and, second, for the designer who wants to extend the features of the framework.

References

Anderson and al. (1999). Seti@home: The search for extraterrestrial intelligence. Technical report, Space Sciences Laboratory, University of California at Berkeley.

Bensaid, N. and Mathieu, P. (1997). A hybrid and hierarchical multi-agent architecture model. In *Proceedings of PAAM'97*, pages 145–155.

Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. (2000). Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66.

Crouch, S. and Starfield (1983). *Boundary Element Methods in solid mechanics*. Georges Allen & Unwin.

[6] `http://www.javasoft.com/products/javawebstart`

Elwyn R. Berlekamp, John H. Conway, R. K. G. (1982). *Winning Ways for your mathematical plays*. Academic Press.

Ferber, J. and Gutknecht, O. (1998). A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of ICMAS'98*.

Ferber, J. and Gutknecht, O. (1999). Operational semantics of a role-based agent architecture. In *Proceedings of ATAL'99*.

Franklin, S. and Grasser, A. (1996). Is it an agent, or just a program?: A taxonomy for autonomous agent. In *Proceedings of the 3rd International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag.

Gray, P. A. and Sunderam, V. S. (1997). IceT: distributed computing and Java. *Concurrency: Practice and Experience*, 9(11):1161–1167.

Gregory, S. G. (1998). xdu: A java-based framework for distributed programming and application interoperability.

Gutknecht O., Ferber J., M. F. (2000). The madkit agent platform architecture. Technical report. `http://www.madkit.org/papers/rr000xx.pdf`.

Harrison, W. (1993). Subject-oriented programming.

Hewitt, C. (1979). Viewing control structures as patterns of passing messages. In *Artificial Intelligence: An MIT Perspective*. MIT Press, Cambridge, Massachusetts.

Philippsen, M. and Zenger, M. (1997). JavaParty — transparent remote objects in Java. *Concurrency: Practice and Experience*, 9(11):1225–1242.

Routier, J. and Mathieu, P. (to appear in April 2002). A multi-agent approach to cooperative work. In *Proceedings of the CADUI'02 Conference*.

Routier, J., Mathieu, P., and Secq, Y. (2001). Dynamic skill learning: A support to agent evolution. In *Proceedings of the AISB'01 Symposium on Adaptive Agents and Multi-Agent Systems*, pages 25–32.

Stankovic, N. and Zhang, K. (1998). Java and network parallel processing. In *PVM/MPI*, pages 239–246.

Wolski, R., Plank, J., Brevik, J., and Bryan, T. (2000). Analyzing market-based resource allocation strategies for the computational grid.

Wooldridge, M. (1997). *Handbook of Agent Technology*, chapter Agent-Based Software Engineering. IEE Proc. on Software Engineering.

Wooldridge, M., Jennings, N., and Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*.

AISB Journal **1**(2), © SSAISB, 2002
http://www.aisb.org.uk

Learning to Colour Greyscale Images

Penousal Machado*, André Dias† and Amílcar Cardoso†

* Instituto Superior de Engenharia de Coimbra, Quinta da Nora, 3030 Coimbra, Portugal, *machado@dei.uc.pt*
† CISUC - Centre for Informatics and Systems, University of Coimbra, Pinhal de Marrocos, 3030 Coimbra, Portugal, *adias@student.dei.uc.pt ; amilcar@dei.uc.pt*

Abstract

This paper is about the colouring of greyscale images. More specifically, we address the problem of learning to colour greyscale images from a set of examples of true colour ones. We employ Genetic Programming to evolve computer programs that take as input the Lightness channel of the training images and output the Hue channel. The best programs evolved can then be used to give colour to greyscale images. Due to the computational complexity of the learning task, we use a genome compiler system, GenCo, specially suited to image processing tasks.

1 Introduction

The work presented here is part of a wider research project, NEvAr, whose aim is to build a constructed artist (i.e. a program that generates artworks autonomously).

NEvAr is an Evolutionary Art Tool inspired on the work of K. Sims (Sims, 1991). It relies on Genetic Programming (GP) (Koza, 1992) to evolve populations of images, based on aesthetic principles. Fitness assignment plays, like in most Evolutionary Computation systems, a key role since it guides the evolutionary process.

NEvAr can be used as an Interactive Evolution tool. In this mode of operation the user supplies the fitness values to the evolved images. It can also be used as a fully autonomous system. In this case fitness is assigned through an explicit fitness function, which takes into consideration several complexity estimates of the images (Machado and Cardoso, 2002).

However, the fitness assignment procedure only takes into account the lightness information of the images, discarding the hue and saturation information. Therefore, in this mode of execution, we are limited to greyscale images. A full description of NEvAr and of the automatic fitness assignment can be found in (Machado and Cardoso, 2002).

There are good theoretical and artistic reasons to deem colour less important than lightness. The development of the colouring procedures of systems like AARON (Cohen, 1995) is, to some extent, based on this notion. However, this collides, at least apparently, with the importance given to colour by some of the most prominent painters (e.g. (Kandinsky, 1911)). Moreover, NEvAr's limitation to greyscale images, in its autonomous version, was frustrating, to say the least. In this paper we address the problem of giving colour to greyscale images.

An analysis of the role of colour and the way colour is assigned, particularly in abstract art, leads to the conclusion that artists (certainly not all, but at least a significant

proportion) usually work with a limited colour palette, and that the spatial relation between colours usually follows some rules. This is consistent with the view that each artist constructs its own artistic language, which complies with an implicit grammar. It is also consistent with the approach used in AARON to colour its drawings (Cohen, 1995; Cohen, 1999).

The idea of creating a program to give colour to the greyscale images created by NEvAr emerged naturally. Unfortunately this poses several problems. Our system is based on a non-symbolic approach and produces bitmap images, hence there is no clear definition of closed forms, shapes, etc.

Therefore, although we could define a palette to work with, assigning the colours of that palette to specific forms would be difficult since we have no forms to begin with. Even assuming that the forms could be properly identified by some sort of pre-processing method, assigning the right colour to each shape and keeping a proper spatial relation among colours would still be a problem, due to the unstructured nature of the output.

Additionally, the creation of a colouring system, by itself, doesn't appear to be an easy task, involving the choice of an adequate set of palettes, establishing a consistent colouring grammar, etc.

Taking these facts into consideration, and also the fact that the generality of this type of approach would be limited, we decided to abandon this idea. Instead, we are trying to create a system that learns to colour images from a set of training examples.

This approach has, potentially, several advantages over a built-in colouring procedure, namely:

- We don't need to code by hand a set of colouring rules.
- The results of the system are less predictable.
- We can use paintings made by well-known artists as training set, hence learning to colour images according to their style.

Furthermore, it's also an indirect way of testing if the colouring procedures followed by some artists can be formally expressed.

In the remainder of the paper we will describe our efforts to build a system to colour greyscale images. The paper is structured as follows: in Section 2 we describe our approach, giving the necessary implementation details; in Section 3, we present some experimental results attained by this approach and make a brief analysis; finally, in Section 4, we will draw some conclusions and present our ideas for future research.

2 Our Approach

GP is one of the most recent Evolutionary Computation techniques. Its goal is to evolve populations of computer programs, which improve automatically as evolution progresses (Banzhaf et al., 1998).

Due to the outstanding influence of the work of Koza (Koza, 1992) it is common, within the Machine Learning community, to associate the term GP to the evolution of tree structures. In this paper we follow this "classical" definition. Therefore, when we talk about GP we are talking about the evolution of tree structures, which are built from a set of functions (f-set) and terminals (t-set). The internal nodes of the tree are members of the f-set, and the leafs are members of the t-set.

In our approach we use GP to evolve populations of programs that give colour to greyscale images. We start by selecting a true-colour training image (or set of images), which is split in its Lightness, Hue and Saturation channels.

The evolved programs take the greyscale image corresponding to the Lightness channel as input, and output a greyscale image. The output is compared with the Hue channel of the training image, the closer the output is to the desired one, the higher the fitness. The same procedure can be applied using the Saturation channel, to evolve programs that generate the saturation information.

As evolution progresses, the quality of the individuals increases and, eventually, we find programs which generate colourings very close to the original ones.

It is important to notice, however, that this is not enough - the idea is to use these programs to give colour to different images. The fact that a program produces an output that exactly matches the training one does not guarantee that it will produce an interesting colouring on different images. In other words, the evolved programs must generalise well. To promote their generalising capabilities, we took some precautions in the selection of the function set and also in the construction of the fitness function. In the remainder of this section, we describe the options taken and give justification for these options.

2.1 Implementation details

A first word goes to how the output of each program is calculated: given a particular individual, it is run for each of the pixels belonging to the training image (or images). Therefore, assuming a training image of 100*100 pixels, each individual must be run 10000 times. Each execution of an individual implies the transversal of its tree and calling, for each node, the corresponding function.

When we take into account that the individuals can easily reach sizes of several thousand nodes, and that GP populations usually contain several hundred individuals, the conclusion that the execution step is of considerable computational weight clearly follows. In order to minimize this problem we implemented our system with GenCo, a Genome Compiler system specially suited for image processing tasks (Dias et al., 2002).

A Genome Compiler is a GP system that makes online compilation of the evolved programs. In situations like the one previously described, i.e. in which each individual must be executed several times, this type of system can provide significant speed improvements, since each individual is compiled once and the resulting machine code executed several times (10000 considering a training image of 100*100 pixels). In this scenario, genome compiler systems are, typically, 50 to 100 times faster than standard C based GP implementations (Fukunaga et al., 1998; Nordin, 1994; Nordin and Banzhaf, 1995; Dias et al., 2002).

Next we describe the design options made in the selection of the function and terminal set, and the reasons that justified them.

2.1.1 Function and Terminal Sets

Our problem has some similarities with the symbolic regression of functions or images (Koza, 1992; Nordin and Banzhaf, 1995). There are, however, some important differences. In a symbolic regression task, one would usually resort to a function set composed by the arithmetic operations and the if statement, and use as terminal set the variables X, Y. This type of set-up unavoidably results in programs whose output depends exclusively on the coordinates of the pixel being calculated.

In an attempt to solve this problem, we added to the terminal set the lightness value of the pixel being evaluated, thus giving more information to the program. This allows the output to vary in accordance to changes on the lightness channel. Additionally, we also added the lightness values of the adjacent pixels, so that the programs have access to the surrounding context of each pixel.

In what concerns the function set, we decided to keep the "traditional" one. The reason for this choice is threefold: it was deemed sufficient for a first approach; the inclusion of more complex functions (e.g. for-next loops, or high level constructs) would severely increase the computational weight of the evaluation step; the inclusion of this type of functions doesn't necessarily benefit the evolutionary process, in fact, the opposite can, and frequently happens (see, e.g., (Banzhaf et al., 1998)).

Taking all these factors in consideration we used the following function and terminal sets:

- F-set = $\{+, -, *, \%, \sin, \cos, if\}$,where $\%$ stands for the protected division operator, and if for the if-less-then-else statement (Koza, 1992).
- T-set = $\{X, Y, A..I\}$,where X, Y are variables corresponding to the coordinates of the pixel, and $A..I$ are the lightness values of the pixel being calculated and of the surrounding ones.

The results achieved were very disappointing, basically because the behaviour of programs continued to be determined, almost exclusively, by the values of the X and Y variables. This is clearly undesirable, since it means that we are not evolving programs that give colour to images according to their lightness channel, we are merely evolving a function that, when applied over an interval of $\mathbf{X}$, $\mathbf{Y}$ values, generates the hue channel of the training image. Thus, we are only memorizing the training instance(s).

Taking the X, Y variables from the terminal set resulted in having poor evolution, and convergence to trivial and uninteresting colourings (e.g. having as a result the predominant colour of the training image, or always assigning to a specific lightness value the same hue or saturation).

It was clear that the variables X, Y were not producing any desirable impact on the programs. Therefore, we decided to delete them from the terminal set. It was also clear that without these variables the programs hadn't enough information to determine the appropriate colour for each pixel, since the programs only have a local view of the lightness channel (the pixel being evaluated and the nine surrounding ones). To compensate this lack of information, we introduced a new function, $get(\Delta x, \Delta y)$, whose behaviour can be described as follows: assuming that the current pixel has the coordinates a, b, it returns the lightness value of the pixel situated on $(a + \Delta x, b + \Delta y)$.

Since the get function provides a way to access the lightness values of the surrounding pixels, there was no reason to make these values as part of the terminal set. Accordingly, our function and terminal sets became:

- F-set = $\{+, -, *, \%, \sin, \cos, if, get\}$
- T-set = $\{E\}$, where E stands for the lightness value of the pixel being evaluated.

This set-up proved to be adequate, allowing the evolution of suitable colouring programs with good generalization capabilities, as will be shown in section 3. In the following section we focus on the used fitness functions, and on the grounds for using them.

2.1.2 Fitness Functions

We start by describing the fitness function used when we are evolving the saturation channel of an image from the lightness one. In this situation we use the root mean square error between the desired output and the real one to assign fitness, i.e. considering that I holds the desired saturation values and that P is the output of a program the fitness is given by the following formula:

$$s = \frac{1}{1 + \frac{\sqrt{\sum_{x=1}^{\max_x} \sum_{y=1}^{\max_y} (I(x,y) - P(x,y))^2}}{\max_x \times \max_y}} \tag{1}$$

When we are trying to evolve the Hue information, this formula must be slightly altered due to the circular nature of the Hue channel. Assuming that the images take values in the $[0, 1]$ interval, the above formula yields a maximum distance when $I(x, y) = 1$ and $P(x, y) = 0$ (or vice-versa). However, in this situation the difference in hue would be quite small and probably unnoticeable to a human observer. Therefore, we use the following formula:

$$h_a = \frac{1}{1 + \frac{\sqrt{\sum_{x=1}^{\max_x} \sum_{y=1}^{\max_y} [\min_\theta (I(x,y), P(x,y))]^2}}{\max_x \times \max_y}} \tag{2}$$

where I holds the desired Hue values, P the program's output, and $min_\theta(x, y)$ returns the minimum angle distance between x and y. This fitness function can be further improved if we take into consideration that the perceived difference in hue depends on the lightness of the pixel (e.g. if the pixel as lightness equal to zero it will always be black, no matter what hue we assign to it; even when the lightness is only close to zero, giving to that pixel a hue different than the desired one will hardly be noticeable to a human viewer). Taking this into account, and considering that L holds the lightness information we obtain the following formula:

$$h_{a'} = \frac{1}{1 + \frac{\sqrt{\sum_{x=1}^{\max_x} \sum_{y=1}^{\max_y} [\min_\theta (I(x,y), P(x,y)) \times (1 - 2 \times |0.5 - L(x,y)|)]^2}}{\max_x \times \max_y}} \tag{3}$$

The tests conducted using formulas 2 and 3 as fitness functions yield deceptively good results. In the early steps of evolution fitness increases steadily and swiftly, giving the impression that an adequate colouring program will be easily found. However, after a few hundred generations, the improvements in fitness drop suddenly and evolution seems to halt. This wouldn't be a cause for concern if the evolved programs solved the problem at hand satisfactorily. To some extent they do, since they usually have high fitness values, relatively close to the maximum attainable fitness. The problem is that a qualitative analysis of the results reveals that the colourings are usually uninteresting from an aesthetic perspective - typically, only a small subset of the original colours is used, resulting in the loss of nuances that make the original colouring work. This situation becomes more severe when the training images have a large area filled with a particular hue value, and other areas filled with hue values close to that one.

To better explain the problem we will use an example: consider, for instance, a landscape painting mostly filled with green (for the grass and trees) and with a blue sky (green

and blue are relatively close in the colour spectrum); a program that outputs the hue corresponding to the green colour will have a relatively good fitness, since it is not penalised in the dominant green area of the image and only suffers a little penalisation on the blue area. To make things even worse, such a program is easy to evolve, so it will be found in a small number of generations. Changing it by mutation or crossover will tend to decrease its fitness, meaning that the fittest descendants will tend to be similar to it. Therefore, in a few generations the population will be dominated by similar programs, which will begin to increase in their size to protect themselves from destructive crossover and mutation; soon the increase in size becomes exponential and evolution becomes impossible (the exponential growth of program size is usually called bloat problem; a more detailed explanation of why bloat occurs can be found in (Banzhaf et al., 1997).

In an attempt to force our system to evolve more interesting colourings, covering a wider colour spectrum, we decided to add another factor to our fitness function. A description of the procedure used to calculate that factor follows.

We start by taking the I image (that holds the desired hue values) and decrease its colour depth, using the optimised median cut algorithm, obtaining an image I' composed by only 16 different hue values $(v_1..v_{16})$. Then we count how many pixels exist of each different hue and store the pixel count values in an array, $A_{I'}$. For each pixel of image P, which holds the output of the program, we determine the closest hue value, v_i, and the distance Δv between the pixel value and v_i. We add to $A_{P'}[i]$ the value $1-\Delta v$, thus if the pixel's hue matches exactly one of the sixteen hues present in the training image we add one, when it doesn't match exactly we add slightly less. After performing this procedure for all the pixels of the output image we compute the following formula:

$$h_b = \frac{1}{1 + \frac{\sqrt{\sum_{x=0}^{15}(A_{I'}[i]-A_{P'}[i])^2}}{16}} \tag{4}$$

Thus, we are basically comparing the number of pixels of each hue value of the output and target image. Returning to our previous example of the green and blue landscape, if the output image has the same amount of blue pixels and green pixels than the original (and also assuming that it is the exact blue and green tone), formula 4 will give a value close to one. However, if the output image is dominated by the green colour there will be a huge discrepancy between the amount of green and blue of the two images, and therefore H_b will be close to zero. Notice that, in what H_b is concerned, the placement of the colours has no real influence on the resulting value. To further force a wider coverage of the colour spectrum, we also used the following modification to this formula:

$$h_{b'} = \frac{1}{1 + \frac{\sqrt{\frac{\sum_{x=0}^{15}(A_{I'}[i]-A_{P'}[i])^2}{\max(A_{I'}[i],A_{P'}[i])}}}{16}} \tag{5}$$

which ensures that all sixteen different hues have the same weight in the calculation. Resorting again to our example, and considering that we have a small yellow area (for the sun), when we use formula 5 having the correct number of yellow pixels is as important as having the correct number of blue or green ones. In the next section we will present some of the experimental results achieved.

3 Experimental Results

In order to test our approach to the colouring of greyscale images we conducted a series of experiments. As training images we selected some of the early works of Wassily Kandinsky. These images were reduced to a size of 96*96 pixels in order to allow a reasonably fast evolution. In its current form our system is only capable of handling square images, so the aspect ratio of the original pictures was altered in order to comply to this constraint. The criteria for selection of the training images was, mainly, individual taste, but we also took into consideration the complexity of the image and the original aspect ratio, to ensure that the images would still be recognizable after being submitted to the previously described operations.

Although our system is prepared to use several training images at the same time, we decided to use, in each experiment, just one training image, since this allowed faster testing and still gives a good indication of the potential of the system.The results presented in this section concern the evolution of programs that take the lightness channel of an image as input and generate the hue channel.

The experimental settings were the following:

- Population size = 100; Tournament selection, Tournament size = 5; Initialisation method = Ramp half and half; Initial tree depth = 2-6, Maximum tree depth = 20; Crossover rate = 0.7, Mutation rate = 0.25, Reproduction rate = 0.05;

The runs were stopped after 1000 generations. We used $h_{a'} + h_{b'}$ as fitness function, since preliminary testing indicated that this type of fitness yield better results.

The best way of presenting the results of our system would be to show the colourings that it generates. Due to the impossibility of presenting true-colour images, we decided to make the figures that should be included in the paper available at the following address:

- http://www.aisb.org.uk/aisbj/extra/1/2/learningtocolour.pdf

In the remainder of this section we will make a qualitative analysis of the results achieved. To get a clear view of the system's strengths and weakness the reader should visit the indicated address.

In the appraisal of the results we must take two different aspects into consideration , namely: 1) How well do the evolved programs colour the training image; 2) How do they behave when applied to different images.

In what concerns to the first criterion, an analysis of the results shows that the generated colourings match closely, although not exactly, the colourings of the training images.

Not having exact matches isn't cause for concern. In fact, we were not expecting to, nor particularly interested in, achieving them, since our primary goal is to obtain programs that generalise well.

In the experimental results present here the maximum number of generations was set to 1000. We did, however, conduct some experiments in which this number was significantly higher. These tests showed that fitness continued to increase steadily (though slower). However, after 50000 generations the exact match was still to be found, and at this point, the size of the programs had increased enormously (the average size was close to 15000 nodes) and the rate of fitness increase was extremely slow.

A careful analysis of the generated colourings will also reveal that, in some cases, there are small areas of noise. These noise areas considerably affect the quality of colourings, since it is possible for a human observer to identify these spots, without resorting to a comparison to the original colouring.

The fact that we are applying the programs to images significantly larger than 96*96, contributes to the appearance of noise. There are, however, other factors involved.

We tried to identify in which situations evolution may lead to this kind of results. Apparently, they occur when, in the early stages of the evolutionary process, an individual is found that generates a reasonable colouring with some noise areas. Since the evolutionary process is in the beginning, the average fitness is low, and the individual will have a fitness value higher than average, and hence great chances of generating descendants. When the areas of noise are relatively small, they will not hinder significantly the individual's fitness. In these circumstances evolution will tend to improve other areas of the drawing (e.g. try to match exactly the hue of a certain area). The problem is that, after a few generations, the portions of code generating noise will be, not only hidden in the middle of the pieces of code that generate a positive effect on fitness, but, probably, also mixed with them. Thus, it becomes extremely difficult to eliminate the undesirable pieces of code.

This type of problem is not particular to our approach. Both natural and artificial evolutionary systems show this type of shortcoming. In our particular case, the best way to avoid it would be to introduce a penalty for the individuals that generate noise areas. This involves, off course, implementing a procedure to identify these noise areas. Although, we still haven't tried it, we feel that the application of a median filter to the target and generated hue channels, followed by a comparison of the changes introduced by it, should be enough to identify most of the noise areas.

More important than the performance of the programs when applied to the training, is their performance when used to colour images that were not use in their training. After all, this was the original goal of our research.

We weren't, obviously, expecting the programs to reproduce the colourings of images that weren't used in their training. Considering that in each experiment only one training image was used, that could only occur by sheer luck.

Like we stated before, the original motivation for our work was to evolve programs that could be later used to colour the greyscale images generated by NEvAr. Therefore, what we wanted to verify was if the evolved programs could generate consistent, appealing and believable colourings.

The experimental results indicate that this is the case. Some of the colourings generated by the programs are quite striking. Moreover, they present a good and (at least in our opinion) aesthetically sound combination of colours.

It's also interesting to notice that, in some cases, the generated images are surprisingly close to the original ones. Like we stated before we were not expecting for this to happen.

We consider these results to be extremely promising, especially if we take into consideration that there is still lot's of room for improvement. To allow a more equitable assessment of the results, avoiding the bias induced by already knowing the original image, we decided to present three alternative colourings of Wassily Kandinsky painting "Interior (My Dining Room)", without presenting the original painting.

A final word goes to the fact that, in some cases, the generated colourings exhibit some of the already identified illnesses, the presence of noise areas. To some extent, this was expected, since the problem lies somewhere in the individuals' code. The application of the programs to images other than the training one tends to make these problems more visible, which was also foreseeable.

4 Conclusions and further work

In this paper we presented our ongoing research whose goal is to learn to colour greyscale images from a set of training instances. This effort is part of a wider research project that aims at building a fully autonomous constructed artist.

The results achieved so far concern, mostly, the evolution of programs that generate the hue channel from the lightness one. The experiments performed on the evolution of programs to generate the saturation channel seem to indicate that this task is quite simpler. Nevertheless, the replacement of the original saturation channel by one generated through a program will unavoidably imply the introduction of further noise. At the time of writing we can't access exactly how much this will affect the quality of the generated colourings (although we foresee little impact).

There is also the need to conduct additional experiments, especially to use several training images simultaneously, which will hopefully increase the generalization capabilities of the evolved programs. Another aspect that can be improved is the fitness function, we are currently altering it so that it also takes into account the vicinity relations between areas of colour, once again the idea is to promote the evolution of programs that assign colour based in more general concepts.

A final word goes to other types of application of the proposed techniques, for instance, the colouring of black and white movies. Assuming that one of the frames is coloured (either by hand or by some other method) it should be possible to evolve a program that gives correct colours to the surrounding frames.

Acknowledgements

This research project was approved by FCT (Fundação para a Ciência e Tecnologia) and POSI (Programa Operacional "Sociedade da Informação"), and is partially funded by FEDER, project n. POSI/34756/SRI/2000.

References

Banzhaf, W., Nordin, P., and Francone, F. D. (1997). Why introns in genetic programming grow exponentially. In *Workshop on Exploring Non-coding Segments and Genetics-based Encodings, ICGA*, East Lansing, MI, USA.

Banzhaf, W., Nordin, P., Keller, E., and Francone, F. D. (1998). *Genetic Programming - An Introduction*. Morgan Kaufman.

Cohen, H. (1995). The further exploits of aaron, painter. *Constructions of the Mind*, 4(2).

Cohen, H. (1999). Colouring without seeing: a problem in machine creativity. *AISB Quarterly*, (102):26–35.

Dias, A., Machado, P., and Cardoso, A. (2002). Improving genome compiler's performance. In *Genetic and Evolutionary Computation Conference, Graduate Student Workshop, GECCO'02*, New York, USA.

Fukunaga, A., Stechert, A., and Mutz, D. (1998). A genome compiler for high performance genetic programming. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 86–94.

Kandinsky, W. (1911). *On the Spiritual in Art*. Republished by Dover Publications in 1997.

Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Machado, P. and Cardoso, A. (2002). All the truth about nevar. *Applied Intelligence, Special issue on Creative Systems*, 16(2):101–119.

Nordin, P. (1994). A compiling genetic programming system that directly manipulates the machine-code. In Kenneth, E., editor, *Advances in Genetic Programming*, pages 311–331. MIT Press.

Nordin, P. and Banzhaf, W. (1995). Complexity compression and evolution. In *Sixth International Conference on Genetic Algorithms, ICGA95*, pages 310–317. Morgan Kaufmann.

Sims, K. (1991). Artificial evolution for computer graphics. *ACM Computer Graphics*, 25:319–328.

AISB Journal **1**(2), © SSAISB, 2002
http://www.aisb.org.uk

From Virtual Bodies to Believable Characters

Marco Vala, Ana Paiva and Mário Rui Gomes

IST / INESC-ID
Rua Alves Redol, 9 - 1000-029 Lisboa - Portugal
marco.vala@gaips.inesc.pt ; ana.paiva@gaips.inesc.pt ; mrg@inesc.pt

Abstract

This paper presents an approach for creating believable characters. We use predefined animations and body postures that can be combined in real-time to generate a rich set of behaviours. Moreover, parameters like speed or spatial amplitude can also be modified in real-time to influence the character's movement. Our ultimate goal is to create reusable synthetic characters that are able to express their inner-feelings using bodily expression.

1 Introduction

The last decade witnessed an impressive evolution of synthetic characters. The recent advances in technology, particularly the continuous increase of performance in computers, lead to broad use of synthetic characters in several areas. Computer simulations are a reality in many domains from warfare to rescue training. The use of synthetic characters allows safe exercises without putting anyone into harm. In education, synthetic characters have been used as tutors that are able to explain or guide a student through a task. Characters can also act as team-mates for individual training in tasks that need cooperation. In the cinema, the movie "Toy Story" opened the door to a completely new generation of movies where the actors are computer generated. Recent examples are "Shrek" and "Final Fantasy". Computer games bring us every day a full hand of new gaming experiences where synthetic characters assume the main role. As a brief example, look at "Tomb Raider" and the impact of a character like "Lara Croft", or the amusing "Guybrush Threepwood" from "The Escape from Monkey Island". Although purely fictional, these characters have a personality, likes and dislikes, friends, and many other things that drag us into the story and make us feel as a part of it. Computer games have also benefited with the development of accurate characters that simulate the reality, especially in sport simulations. Games like "FIFA" or "NBA Live" have models for most of the players and it is quite pleasant to recognise a star like "Figo" or "Kobe Bryant". Likewise, many synthetic actors start to participate in artistic performances or TV shows, and start to play an important role in these domains. The future platforms of interactive TV and TV on demand will offer a "workplace" for many virtual presenters, virtual entertainers and virtual advertisers. We are entering a new era where synthetic characters need to be more human-like and, above all, more believable.

The body assumes a natural relevance in this new generation synthetic characters. Humans, for instance, express their emotions and inner feelings with facial expressions, but also with gestures and behaviours that affect the body movement. If we take a close

look into the roots of animation, we quickly conclude that skilled animators use the body and the way it moves to denote personality, express emotions and appear to have a certain inner life, which is sufficient to induce believability. They are able to create characters that easily delude our eyes. Therefore, a synthetic character should be able to express its emotions through bodily animation using the appropriate gestures and behaviours.

The overall goal of our work is to provide a simple way of creating and controlling reusable bodies of synthetic characters that can express emotions. That is, to develop reusable synthetic characters with an expressive bodily behaviour.

2 Background

Traditionally, synthetic characters were created using a pure computer graphics approach in which the visual realism is the ultimate goal. Different researchers, such as (Badler et al., 1993), (Fua et al., 1998), (Kalra et al., 1998), and (Aubel and Thalmann, 2000) look at believability as a strict visual problem. The approach is based on detailed geometrical models and advanced animation techniques that are able to assure a good visual accuracy. However, the generated characters and movements are computational expensive, and the results are often unrealistic. Characters lack a certain inner life that is essential to delude our eyes. Consequently, the idea that believability depends more on the characters' ability of conveying its inner-feelings lead to a different approach that relegates the visual realism to a second plane. Blumberg and Galyean in (Blumberg and Galyean, 1995), Perlin and Galyean in (Perlin and Goldberg, 1996), Russell and Blumberg in (Russell and Blumberg, 1999), and Johnson et al. in (Johnson et al., 1999) seek what can be called behavioural realism, and look at high-level architectures for real-time animation and interactive control. The generated characters are more expressive and more alive, but that richness is also very dependable on the animators' ability to maintain a certain behavioural coherency in the characters' library of pre-defined movements. Therefore, the characters can only express the feelings or the emotions that were previously modelled, and most of the animations cannot be reused in different characters.

In parallel, several researchers tried to find a way to change the movements in real-time to add expressiveness, personality or emotions. Some approaches propose secondary motions as a way to add naturalness to primary motions, like Perlin's work that uses periodic noise functions to add expressiveness (Perlin, 1995). Others used signal analysis techniques to capture and create new motions: Unuma et al. introduce a model to describe and manipulate human periodic motions based on a Fourier Functional Model (Unuma et al., 1995); and Amaya et. al. capture the difference between neutral and emotional motions, and allow the generation of new emotional motions (Amaya et al., 1996). More recently, results from movement observation have been used to define models that parameterise movements in real-time to achieve a rich set of variations: the EMOTE system proposed by Chi et al. uses results from the Laban Movement Analysis to modify arm and torso animations into more expressive movements (Chi et al., 2000) (Badler et al., 2000). However, all these approaches are still very limited and, most of the times, they are only applicable to certain body parts or to a single class of movements.

3 Our Approach

Our work is inspired mainly on Blumberg and Perlin's work, but uses some of the results of the other approaches, namely the Amaya's work on emotional transforms and how the

speed and the spatial amplitude of the movement vary noticeably with different emotions.

The main idea is to use a set of pre-stored movements and modify them in real-time to reflect inner-feelings or emotions. These changes can result from the combination with specific body postures or secondary motions, or from variations of basic parameters like speed or spatial amplitude. For example, imagine a neutral walking movement that we want to modify in real-time to express sadness. The character can begin walking slowly and with smaller steps (and that denotes a variation in the speed and in the spatial amplitude of the movement), the torso could bend to the front (and that can be done by combining the movement with a body posture in which the torso is bent), and the head could occasionally turn to the left and the right alternately denoting a certain disappointment (and that can be achieved using a secondary motion that drives the head).

Moreover, the movement combinations and the variation in the speed and in the spatial amplitude can be parameterised to express different degrees of sadness. For instance, and recovering the above example, we could denote more or less sadness generating a movement more or less exaggerated. Thus, if the character is very sad, the walking movement should be very slow, with very small steps, the torso highly bent and frequent head movements. On the other hand, if the character is only a little bit sad, the walking movement could be moderately slow, with small steps, the torso slightly bent and rare head movements.

4 Architecture

The system has a three-layer architecture as depicted in Figure 1. The behaviour engine is responsible for the high-level control of the character, and it is typically implemented by the character's mind. It sends requests to the animation engine for activation/deactivation of animations or body postures, and for increasing/decreasing the speed and spatial amplitude. The animation engine is responsible for combining all the active animations and implements a simple re-source mechanism to avoid unwanted behaviours. It has direct control over the character's animations and body postures. Finally, the graphics engine is responsible for maintaining the geometric model and for controlling the rendering process.

4.1 Character Model

A character is defined by a geometric model and by a set of pre-defined animations and postures that can be combined in real-time as described above. We use a simple geometric model composed by a hierarchical joint skeleton (Figure 2[1]) and a deformable skin layer attached to the joints (Figure 3[1]). A posture reflects a certain joint configuration and an animation is the variation of the joints or a skin deformation over the time. Using this information, the animation engine is capable of calculating the contribution of multiple active animations and/or postures, and updates the geometrical model every cycle to reflect the desired behaviour.

[1]The colour figures referred to in this paper may be found at
`http://www.aisb.org.uk/aisbj/extra/1/2/VirtualBodies.pdf`.

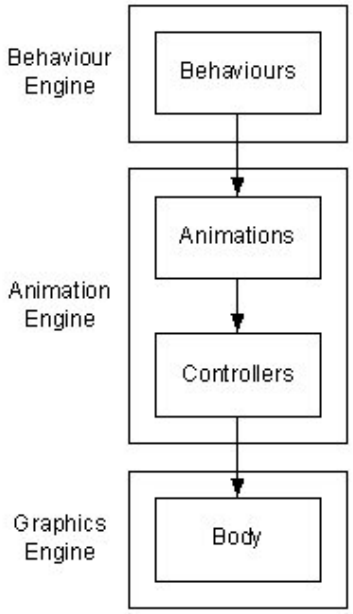


Figure 1: Architecture of the System

4.2 Blending Animations

Typically, applications that use synthetic characters use libraries of animations that are mutually exclusive. For instance, characters used in computer games usually perform one action at a time, and that can be simply achieved using the appropriate animation for each action. However, these characters are often repetitive and predictable, unless you have a huge database of animations.

Our work combines animations and/or body postures in real-time to generate new movements that add a certain behavioural richness to the character. Basically, the animation engine generates a mixed movement of each active animation/posture using a linear interpolation algorithm. Each animation or posture has an associated weight that changes in real-time and that determines the contribution of that specific animation or posture to the generated movement.

This behavioural richness allows us to influence basic animations and create variations that otherwise would have to be created as independent animations. A good example is a ”walk movement” and the possible body posture variations to reflect different emotional states.

4.3 Speed and Spatial Amplitude

Amaya et al. in (Amaya et al., 1996) presented a work where they concluded that the speed and the spatial amplitude of the movement vary noticeably with different emotions. For instance, “sad” movements are normally slow and narrow whereas “happy” movements are fast and wide. Our work uses these results and allows variations in the speed (increase/decrease the number of frames per unit of time) and in the joint amplitude.

Therefore, it is possible to parameterise an animation in real-time to denote inner-feelings or even personality. For example, a lazy person runs more slowly (Figure 4[2]) than an energetic one (Figure 5[2]).

5 Concluding Remarks

Our work does not intend to establish any mapping between emotional states and movements or movement changes. We intend to provide characters that have the ability of modifying their movements in real-time and, therefore, that are able to express emotions. How a particular emotion is expressed is out of the work's scope.

Therefore, the main contribution of the work to the research of expressive characters for social interactions is a simple method of manipulating the character's body to achieve more natural and believable movements. The most immediate advantage is the generation of a rich set of animations using a compact database of pre-defined animations. For instance, it will be no longer necessary to create a "normal" walk, a "sad" walk, a "happy" walk, or a "tired" walk. Using a neutral walk and the right set of parameterisations and body postures we can achieve new animations similar to those produced offline. This allows a smaller dependence on the designers, and reduces the time necessary to create a character and its library of animations.

Another advantage is the possibility of visually identifying generic parameterisations for expressing a specific emotion, which could lead to the development of a broader model that establishes a matching between a theory of emotions and the parameters that affect the animations.

Acknowledgements

This work has been partially supported by the EU funded SAFIRA project number IST-1999-11683. Ana Paiva has also been partially supported by the POSI programme (do Quadro Comunitário de Apoio III).

References

Amaya, K., Bruderlin, A., and Calvert, T. (1996). Emotion from motion. In *Graphics Interface*, pages 222–229.

Aubel, A. and Thalmann, D. (2000). Realist deformation of human body shapes. In *Proceedings of Computer Animation and Simulation*, pages 125–135.

Badler, N., Costa, M., Zhao, L., and Chi, D. (2000). To gesture or not to gesture: What is the question? In *Proceedings of Computer Graphics International*, pages 3–9.

Badler, N., Phillips, C., and Webber, B. (1993). *Simulating Humans: Computer Graphics Animation and Control*. Oxford University Press.

Blumberg, B. and Galyean, T. (1995). Multi-level direction of autonomous creatures for real-time virtual environments. In *Proceedings of SIGGRAPH'95*, pages 47–54.

[2]The colour figures referred to in this paper may be found at
`http://www.aisb.org.uk/aisbj/extra/1/2/VirtualBodies.pdf`.

Chi, D., Costa, M., Zhao, L., and Badler, N. (2000). The emote model for effort and shape. In *Proceedings of SIGGRAPH'2000*, pages 173–182.

Fua, P., Plankers, R., and Thalmann, D. (1998). Realistic human body modeling. In *Fifth International Symposium on the 3D Analysis of Human Movement*.

Johnson, M., Wilson, A., Blumberg, B., Kline, C., and Bobick, A. (1999). Sympathetic interfaces: Using a plush toy to direct synthetic characters. In *Proceedings of CHI'99*.

Kalra, P., Magnenat-Thalmann, N., Moccozet, L., Sannier, G., Aubel, A., and Thalmann, D. (1998). Real-time animation of realistic virtual humans. *IEEE Computer Graphics and Applications*, 18(3):42–56.

Perlin, K. (1995). Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15.

Perlin, K. and Goldberg, A. (1996). Improv: A system for scripting interactive actors in virtual worlds. In *Proceedings of SIGGRAPH'96*, pages 205–216.

Russell, K. and Blumberg, B. (1999). Behavior-friendly graphics. In *Proceedings of Computer Graphics International*.

Unuma, M., Anjyo, K., and Takeuchi, R. (1995). Principles for emotion-based human figure animation. In *Proceedings of SIGGRAPH'95*, pages 91–96.