THE TENTH WHITE HOUSE PAPERS
Graduate Research in Cognitive and Computing Sciences at Sussex

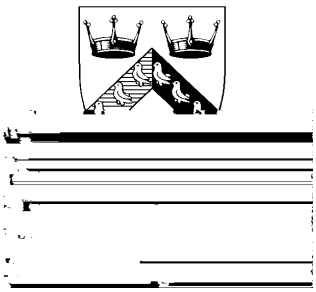
Editors:
John C. Halloran & Fabrice P. Retkowsky

CSRP 478

January 1998

ISSN 1350-3162

UNIVERSITY OF



Cognitive Science Research Papers

THE TENTH WHITE HOUSE PAPERS

Graduate Research in Cognitive and Computing Sciences at Sussex

CSRP 478

Editors:
John C. Halloran & Fabrice P. Retkowsky

January 1998

Contents

Dedication

The editors would like to dedicate the Tenth White House Papers to Sarah Parsowith for all the ‘extras’

Preface

At the Isle of Thorns (a small village situated not far from Haywards Heath), stand a few white buildings. These buildings, which sometimes act as Sussex University's conference centre, are most of the time used as a playground by rabbits. However, every year, they are disturbed by a congregation of COGS research students. In accordance with this time-honoured tradition, 1998 will see the 11th Isle of Thorns workshop, where COGS students will gather to present their work, share some ideas, and spend the rest of the time socialising.

The White House Papers are an introduction to this year's workshop. They include papers written by PhD students in 1997, and shed some light on the diversity of the subjects studied at COGS. They include cognitive psychology, intelligent tutoring systems, logic, medical diagnosis, natural language processing, neural networks, robodu.56 -13(s)-5.52048((o)-4.109142048(,)-210.28e(4422(l)-248.4422(l)-81(e)]TJ24

node of the tree *t* for which it is never the case that $\mathcal{T} \vdash_{\Sigma} \psi$ and $\mathcal{T} \not\vdash_{\Sigma} \psi$. To solve the problem of which defeasible rules should be used to defeat other defeasible rules, Nute provides an explicit partial order "$\sqsupset$" between defeasible rules which is termed as a superiority relation. Thus, if a rule r_1 is ***superior*** to a rule r_2 $(r_1 \sqsupset r_2)$, then it may be used to defeat the ***inferior*** rule r_2. In some cases, it is also possible to extract information about the partial order by using the method of more specific antecedent (***cf. e.g.*** (Nute, 1992)). Defining superiority by antecedent specificity, the antecedent conditions of one rule are derivable from the antecedent conditions of another rule. However, an extra complication is needed, for some parts of the derivations in a proof tree may be based on just the subtheory of $\mathcal{T}$. Therefore, some labellings may vary on their admissible set of literals and rules upon which the proofs of literals in rules defined by more specific antecedents depend.

Different rules interact with each other in the following ways. The rule ψ `:=` Φ is ***defeated*** by the rule η `:=` X or η `:-` X, if the head η is either a negation of ψ or some literal which is explicitly stated as incompatible with ψ. Such a literal η is called ***contrary*** to ψ. Next, the rule ψ `:=` Φ is ***undercut*** by the defeater η `:^` X, if η is contrary to ψ and ψ `:=` Φ is not superior to η `:^` X. Finally, a defeasible rule ψ `:=` Φ or a defeater ψ `:^` Φ may also be ***preempted***, whenever there exists a literal η contrary to ψ which is derivable from the theory, or there is an ordinary Prolog rule η `:-` X with a contrary head, or for the defeasible rule ψ `:=` Φ there is a rule η `:=` X, whose head η is contrary to ψ and which is superior to it, that is, if $(\eta \text{ := } X) \sqsupset (\psi \text{ := } \Phi)$. Furthermore, the literals contained in the bodies X in any of the former rules must always be defeasibly derivable from the defeasible theory $\mathcal{T}$. The role of the preemption is to relax the assumption that applicable defeasible rules to defeat competing contrary rules must always be superior to every other competing rule. When the preemption is enabled, once the defeasible rule is defeated it loses its capacity to defeat any other rule.

We take the latest d-Prolog version (Nute, 1997) as our starting point and show how to extend it with the so called even-if rules and even-if proof conditions given to those rules. Even-if rules and even-if conditions without preemption were given in (Nute, 1994a), but they have not been implemented in d-Prolog. Our task is, thus, to first formulate these conditions in theoretical level such that they also enable preemption, and then show how they can be implemented in d-Prolog. Finally, we shall derive some computational results by computing some sizes of the proof trees for the given examples. Some of the corresponding new d-Prolog predicates are presented in the appendix.

t-iaso13(s)5.28871(t)611(a)5.64534(7(l)6.93181(o)4.1014(n)24.1014(f)4.1014(f)4.1113l)6.92958(e)5.62Tm[(6

 – Let us add to the previous theory a defeater, according to which a Lappman, who has not received EU funds may very well be envious (`((envious(X) :^ (lappman(X), neg funds(X))`). If Tomi is not given funds (`neg funds(tomi)`), it is hard to say, with this amount of information, whether Tomi is envious or not. Perhaps the best thing is to refuse drawing any conclusion. In our extension of d-Prolog, both `neg envious(tomi)` and `envious(tomi)`

(b) *there exists* ψ := $H \in R$ *such that for every* $\eta \in H$ *a node* n *has a child labelled* $\langle K, R, @\eta^+ \rangle$
(some other rule with the same head has a derivable body)

(2) *for every* ψ := Ξ | $\Phi \in R$, *either (a), (b) or (c) holds:*

(a) *there exists* $\varphi \in \Phi$ *and a child of* n *labelled* $\langle K, R, @\varphi^- \rangle$
(a literal in the head is demonstrably not derivable)

(b) *there exists* neg ψ := $X \in R$ *such that for every* $\chi \in X$ *a node* n *has a child labelled* $\langle K, R, @\chi^+ \rangle$
(some contrary rule has a derivable body)

(c) *there exists* neg ψ := Δ | $\Theta \in R$ *or* neg ψ :^ $\Theta \in R$ *such that for every* $\theta \in \Theta$ *a node* n *has a child labelled* $\langle K, R, @\theta^+ \rangle$, *and either (i), (ii) or (iii) holds:*

(i) *there exists* $\theta \in \Theta$ *and a child of* n *labelled* $\langle (\Phi \cup \Xi), R, @\theta^+ \rangle$
(specificity: some literal in the main body of the rule is derivable from the body of the original rule)

(ii) *for every* $\varphi \in \Phi \cup \Xi$ *a node* n *has a child labelled* $\langle \Xi, R, @\varphi^+ \rangle$
(specificity: every literal in the body of the original rule is is derivable from the main body of the contrary rule)

(iii) *there exists* $\varphi \in \Psi \cup \Xi$ *and a child of* n *labelled* $\langle K, R, @\theta^- \rangle$.
(preemption: some literal in the body of the original rule is demonstrably not derivable).

n $\cup \{SS^+, P^+, P^-\}$ *is a defeasible logic.* [1]

PROOF. On the depth of the proof tree t it can be shown that there is no theory $\mathcal{T}$ and a literal ψ such that $\mathcal{T} \vdash \psi$ and $\mathcal{T} \nvdash \psi$. □

3 Extending d-Prolog

To incorporate previous conditions to the latest version of d-Prolog (Nute, 1997), we add to its syntax a new two-place relation "|", which distinguishes between the primary condition Φ in the rule ψ := Ξ | Φ and secondary even-if condition Ξ, as in the previous conditions P^+ and P^-.[2] We need to consider the following modifications and additions to the defeating and undercutting conditions that were given in (Nute, 1997).

The rule ψ := Ξ or ψ :- Ξ is defeated (as implemented in the predicate `defeated/2`), if there exists an even-if rule φ := Ξ | Φ, whose head φ is contrary to ψ, whose body Φ without the even-if condition Ξ is defeasible derivable, and whose even-if condition Ξ is a body of such a rule r which is not superior to the rule φ := Ξ | Φ.

The rule ψ := Ξ | Φ is defeated, (i) if there exists a rule φ := Θ whose head φ is contrary to ψ, Θ is defeasibly derivable, and the condition Ξ is not the head of any defeasible rule, which is not superior to the rule φ := Θ, or (ii) if there exists a rule φ := Φ | Θ, whose head φ is contrary to ψ, Θ is defeasibly derivable, and the even-if condition Φ is the head of a defeasible rule r which is not superior to the rule φ := Φ | Θ.

[1] is a monotonic core of a defeasible logic consisting of the four basic conditions, and $^+$ is a semi-strictness condition. They are both defined, for example, in (Nute, 1992). With the semi-strictness condition, an ordinary rule ψ : Φ may defeat another ordinary rule neg ψ

- The rule ψ `:=` Ξ `|` Θ is undercutted (`undercut/2`), if there exists a defeater φ `:^` Θ, whose head is contrary to ψ, Θ is defeasibly derivable, and ψ `:=` Ξ `|` Θ is not superior to it.

The preemption of defeaters (`preempted/2`) is almost analogical to the defeating conditions (see Appendix A).

There are also two new definitions for the defeasible derivability (`def_der/2`), three definitions for

as possible worlds, for they are the usual epistemic alternatives abreast of possible worlds. However, the truth conditions in impossible worlds are not recursively defined, and hence can be completely free.

The method of impossible worlds semantics works. It solves the problem of logical omniscience for

is defined in the possible worlds semantics, namely, as truth in all possible worlds. For this reason, much stronger variants of the basic semantic framework have been proposed, such as Hintikka-Rantala's impossible worlds semantics (Hintikka, 1975; Rantala, 1982a, 1982b).

An impossible worlds model is a 4-tuple

inadequacy of the original impossible worlds semantics have been put forward by Muskens (Muskens, 1992), who proposes type-theoretic solutions to them.

Choose $\varsigma_1 = \{\varphi, \psi\}$ and $S = \varsigma_1$. Now $\varphi \in \varsigma_1$ and hence $V(B_{a_1}\varphi) = 1$. Similarly, $\psi \in \varsigma_1$ and hence $V(B_a$

basic intuition of knowledge or belief as a truth in all possible worlds, in all epistemic alternatives an agent can conceive, is not well retained. Belief is modelled with the alternative scenarios, but truth is no longer an applicable concept in alternatives. Moreover, beliefs are not even really modelled, but merely represented. It remains to be seen what interesting properties such beliefs have. Furthermore, convincing answers should be tried to find to the questions of whether the two semantics are equivalent and also, where the stipulated descriptive alternatives that represent agents' beliefs come from in the first place.

References

Fagin, R., & Halpern, J. (1988). Belief, awareness and limited reasoning. *Artificial Intelligence*, *34*, 39–76.

Hintikka, J. (1962). *Knowledge and Belief*. Cornell University Press, Ithaca, NY.

Hintikka, J. (1975). Impossible possible worlds vindicated. *Journal of Philosophical Logic*, *4*, 367–379.

	player 2 cooperates	*player 2 defects*
player 1 cooperates	1:R=3 2:R=3	1:S=0 2:T=5
player 1 defects	1:T=5 2:S=0	1:P=1 2:P=1

Table 2: Prisoner's Dilemma Scoring Table

course the logic is the same for your alleged accomplice, and if you both defect then you will both do worse than if you had both cooperated (see table 1).

Cooperation is thus unlikely to arise in a one-shot Prisoners Dilemma, but if players can meet time and time again, and retain some memory of previous interactions, then cooperation on any given move does become a rational strategy. It is this Iterated Prisoner's Dilemma (IPD) that forms the core of the present study.

Many researchers have used genetic algorithms to evolve strategies to play the IPD (see e.g. (Axelrod, 1984),(Langton, 1995)). In these studies, as in the present model, the genotypes comprise of binary character strings representing policies for playing the IPD, with the length of the genotype determining the number of preceding moves (the game history) upon which each individual can base its strategy. It has been repeatedly demonstrated that cooperative strategies can and do arise and persist in artifical ecologies populated by these evolving strategies.

2.2 The Modified IPD

The present paper uses a modified version of the IPD in order to model the interactions between servers and customers in a telecommunications network. The way in which the IPD model reflects the functional constraints of the call admission problem is a major contribution of this paper, but is predicated upon an extension to the basic paradigm that is not unique to the present investigation, concerning *preferential partner selection.*

In the standard formulation of the IPD, interactions are arranged in a very orderly fashion, usually in a 'round-robin' tournament where every individual interacts once with every other on each iteration. The principle of preferential partner selection removes this constraint by allowing individuals to have some control over who they interact with, and this extension can lead to the emergence of interesting new dimensions of emergent behavioural structure.

tomers), with a member of the customer population *not* representing a person with a phone, but instead a distribution node for a set of calls that must obtain access to a server network. These distinct populations interact with each other according to a model derived from th

	server cooperates	*server defects*
type 1 customer coop.	c:R=3 s:R=3	c:S=0 s:T=5
(type 1 customer defects)	c:T=5 s:S=0	c:P=1 s:P=1
type 2 customer coop.	c:R=5 s:R=5	c:S=-2 s:T=8
(type 2 customer defects)	c:T=8 s:S=-2	c:P=-1 s:P=-1

Table 3: Call Admission Model Scoring Table

2.4 Genotype Encoding Scheme

The encoding scheme employed in the present model is an extension of a system employed by Lindgren[4] (Lindgren, 1991).

At the heart of each individual is a genotype, consisting of a string of `c`'s and `d`'s, which determine a strategy for playing the IPD. The longer the genotype, the more it can be influenced by the past history of interactions with other individuals - and the servers maintain separate game histories for each customer that they interact with (and likewise for the customers).

that the customer and server populations do not interbreed. Fitness is determined by the total score on the modified IPD, and it is the genotype coding for the IPD strategy (i.e. the call admission policy) that is modified in the breeding process. Population statistics are also calculated separately for each distinct population.

The main procedure of the model is implemented as follows:

```
randomly initialise bipartite populations
FOR EACH generation
    FOR EACH iteration
        servers fail according to the failure rate
        customers choose most preferable servers and
            make offers (according to variable
            arrival rates)
        servers rate offers and refuse least
            preferable ones that exceed capacity
            (taking account of different service
            requirements in the 2-class problem)
        FOR EACH tolerable offer for each server
            one round of IPD is played (with or
                without noise), and expectations
                are updated
        ENDFOREACH
        game history array and various scores are
            updated, blocking probabilities are
            calculated
        each server status evaluated and brought back
        on line if sufficient down time has elapsed
    ENDFOREACH
    new generations are created through bipartite
        breeding and tournament GAs
    every so often population statistics are
        calculated, and presented with graphics
ENDFOREACH
```

The model was coded in ANSI C and executed on a Sun Sparc workstation.

4 Results

4.1 Overall System Performance

The system typically evolves to very stable situations very quickly, with both customer and server cooperation very near to maximum all the time. This is a promising first observation as it suggests that even when control is completely localised and devolved, and even when short-term benefits can be gained through 'antisocial' behaviour, the system as a whole rapidly reaches an equilibrium beneficial to both customers and servers.

The patterns of connectivity at the beginning of each generation are unpredictable due to the initial expectations all being equal, providing no basis for partner choice. But as each generation wears on, stable patterns of interaction develop, - the same customers being admitted by the same servers, with occasional alterations. This stability persists over many generations.

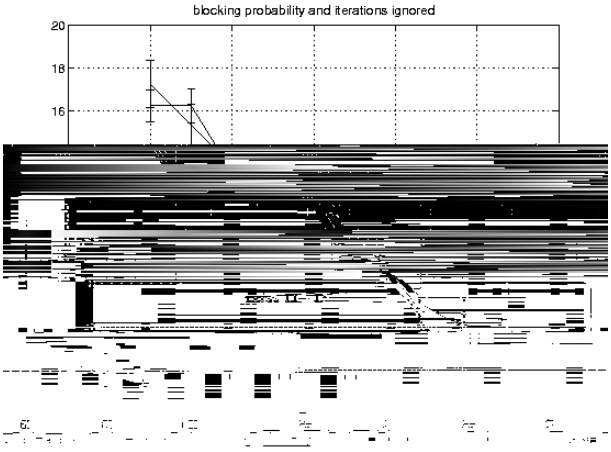


Figure 1: *the blocking rate falls off as each generation progresses (the horizontal scale determines the percentage of the earlier part of the generation that is igonred when calculating the statistic).*

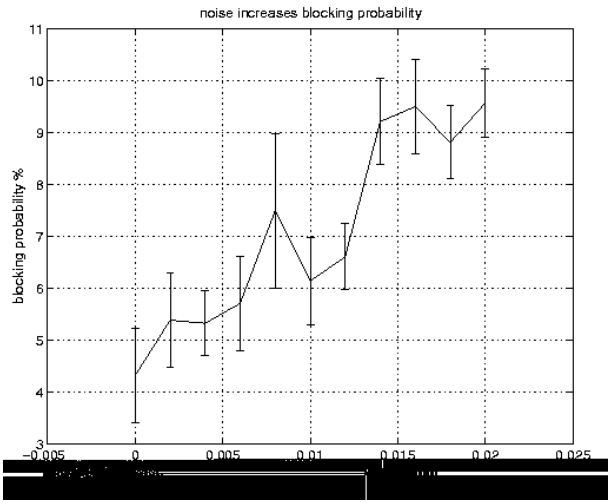


Figure 2:

Figure 3: *low levels of server failure are successfully accommodated by the model, but high levels of server failure lead to system collapse - the critical point is dependent upon server down-time.*

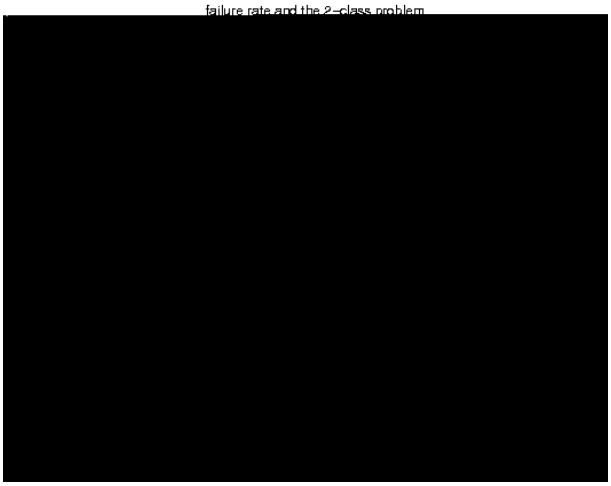


Figure 4: *low priority customers suffer more than high priority customers when servers fail in the 2-class problem.*

4.3 Noise and Failure

The effects of noise and server failure were also explored in the context of the simple problem outlined above, with noise referring to the probability with which the opposite action to that specified by the genotype of the server or customer is performed. In most cases this simply means the probability with which calls are accidentally blocked (although it is also the probability with which blocked calls are accidentally processed). Figure 2 shows that whilst system performance is degraded with increasing noise levels, this degradation is not catastrophic.

Figure 3 examines server failure, demonstrating that with low failure rates the system dynamically reconfigures the flow of call admissions in order to cope - but after a critical stage is reached this process breaks down and almost all calls are blocked. This critical point is dependent upon the server downtime; the longer each server is down for, the earlier the system performance collapses.

4.4 The 2-class Problem

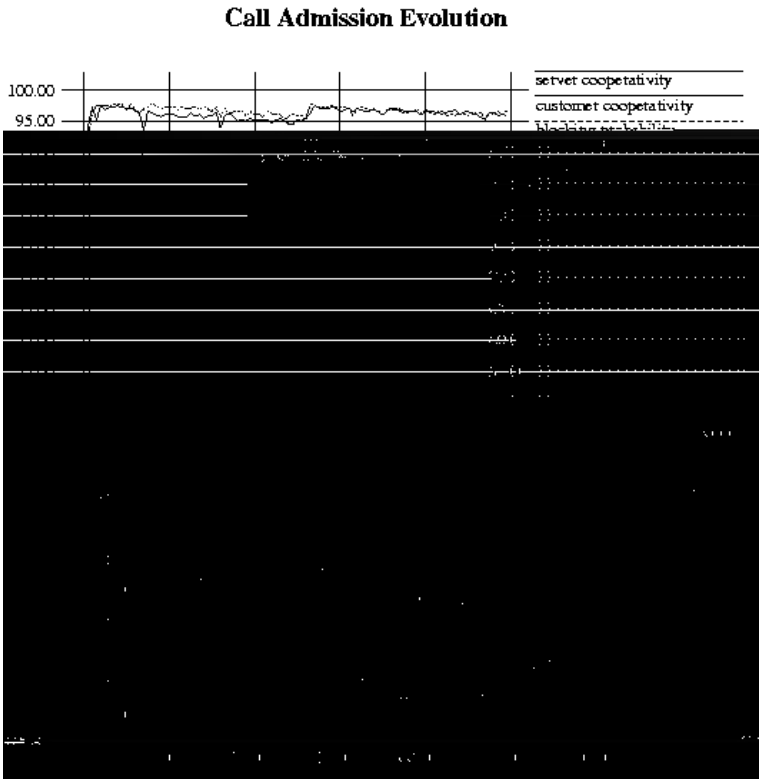


Figure 5: *introduction of extra servers leads to dynamic reconfiguration of the system and a concomitant reduction in the blocking rate.*

However, the introduction of additional servers, in all cases, ameliorated the blocking of these low priority calls.

In sum, the model presented here delivers a new, robust, and effective principle for developing and deploying call admission strategies. The main question which then arises concerns the relationship that this model should have with existing and near-future telecommunications technology. There are two primary alternatives. In principle, the system could be located on-line, with call admission strategies evolving in real-time in a real network. This would be a very s

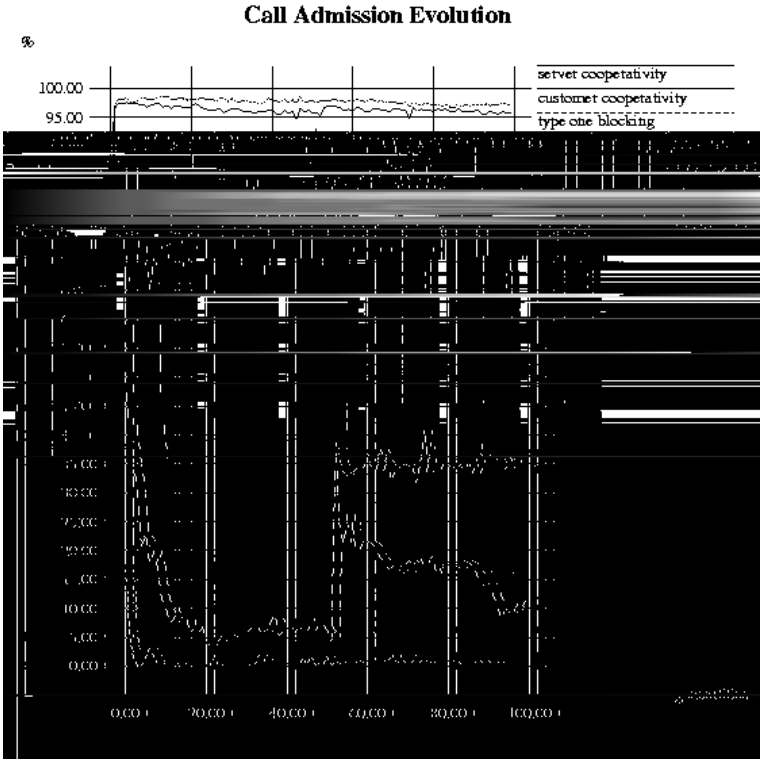


Figure 6: *introducing extra servers reduces the high blocking rate experienced by low priority customers in the 2-class problem.*

Both the system of offers and refusals, and the implementation of the genetic algorithm itself, require the synchronised and centrally co-ordinated interaction of individuals within the system. Practically, this is unlikely to present a problem, and indeed may even be of benefit in that it provides another way in which overall central control can be maintained without diminishing the benefits that accrue from this kind of system.

6 Future Work

6.1 Integration with Executive Control

Integration with a central controller is an obvious avenue f

6.3 Lamarckian Inheritance

Lamarckian inheritance could be introduced into the system, so that expected payoffs are not periodically reset but are passed down from parents to children. This would suggest the use of a continuous GA instead of a generational GA, and would clearly alleviate the problem of the surge in blocking rates at the start of each new generation.

Acknowledgements

This work owes a great deal to my supervisors Phil Husbands and Inman Harvey, and also to Tony White of Nortel Ltd. This work was funded in part by the ESPRC and in part by Nortel Ltd.

References

Ashlock, D., Smucker, M., Stanley, E., & Tesfatsion, L. (1994). Preferential partner selection in an evolutionary study of prisoner's dilemma. Economics report 35, Iowa State University.

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

Langton, C. (1995). *Artificial Life; an Overview*. MIT : Bradford Books.

Lindgren, K. (1991). Evolutionary phenomena in simple dynamics. In Langton, C., Farmer, J., Rasmussen, S., & Taylor, C. (Eds.), *Artificial Life II*. Addison-Wesley.

Littman, M., & Boyan, J. (1993). A distributed reinforcement learning scheme for network routing. In *Proceedings of the First International Workshop on Applications of Neural Networks to Telecommunications*, pp. 45–51.

Magnanti, T., & Wong, R. (1984). Network design and transportation planning: Models and algorithms. *Transportation Science*, *18*, 1–55.

May, R. (1973). *Stability and Complexity in model ecosystems*. Princeton University Press, Princeton, NJ.

Rose, C., & Yates, R. (1996). Genetic algorithms and call admission to telecommunications networks. *Computers and Operations Research*, *23*(5), 485–499.

Stanley, E., D., D. A., & Smucker, M. (1995). Iterated prisoner's dilemma with choice and refusal of partners: Evolutionary results.. In Moran, F., Moreno, A., Merelo, J., & Chacon, P. (Eds.), *Advances in Artificial Life : Lecture Notes in Artificial Intelligence*. Springer-Verlag.

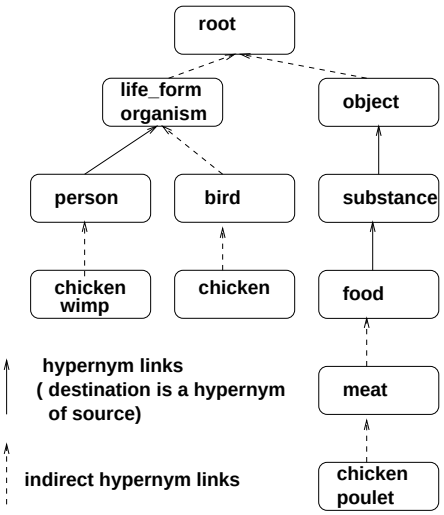


Figure 1: Hypernym hierarchy showing the senses of "chicken".

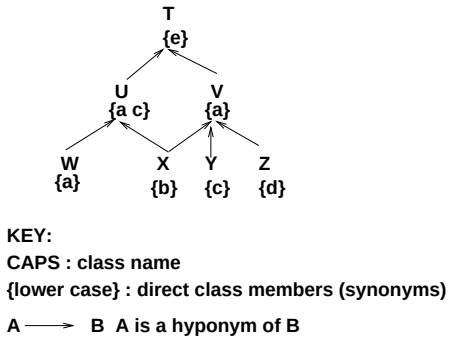


Figure 2: Taxonomy A.

Resnik (Resnik, 1993) and that by Ribas (Ribas, 1995b, 1995a), and Li and Abe (Li & Abe, 1995; Abe & Li, 1996). The description of how the class probabilities are calculated differs between Li and Abe and Ribas but the methods are essentially the same.

To illustrate the difference in the approaches we will consider taxonomy A in figure 2. The lower case letters represent direct membership of the classes (in UPPER CASE) under which they appear. The arrows indicate the hypernymy relationship and any lower case member of a hyponym is an implicit indirect member of the hypernym classes. In this way the class T has *"e"* as a direct member and *"a b c d"* as indirect members. A class may have more than one member e.g. , and an item may belong to more than one class e.g. *"a"*.

Resnik's approach doesn't distinguish between hypernymy and polysemy when estimating the frequency distribution. The frequency of a noun, in a given sample, contributes to all classes the noun belongs to, regardless of direct or indirect membership. Furthermore each frequency count is divided by the number of these classes to ensure that the sum of probabilities over the entire hierarchy equals one, equation 1 describes his estimation of the frequency of a class (c).

$$freq(c) = \sum_{n \in nouns\_at\_or\_under(c)} \frac{1}{|classes(n)|} \times freq(n) \quad (1)$$

Table 4: Resnik's Frequency and Probability Distributions.

Table 5: Ribas' Frequency and Probability Distributions.

CLASS	FREQ	

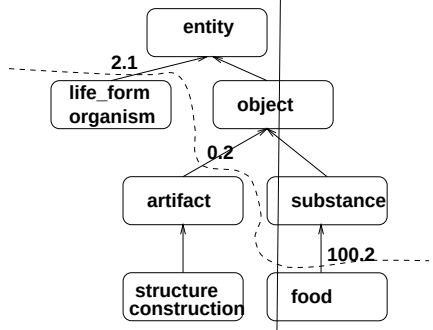
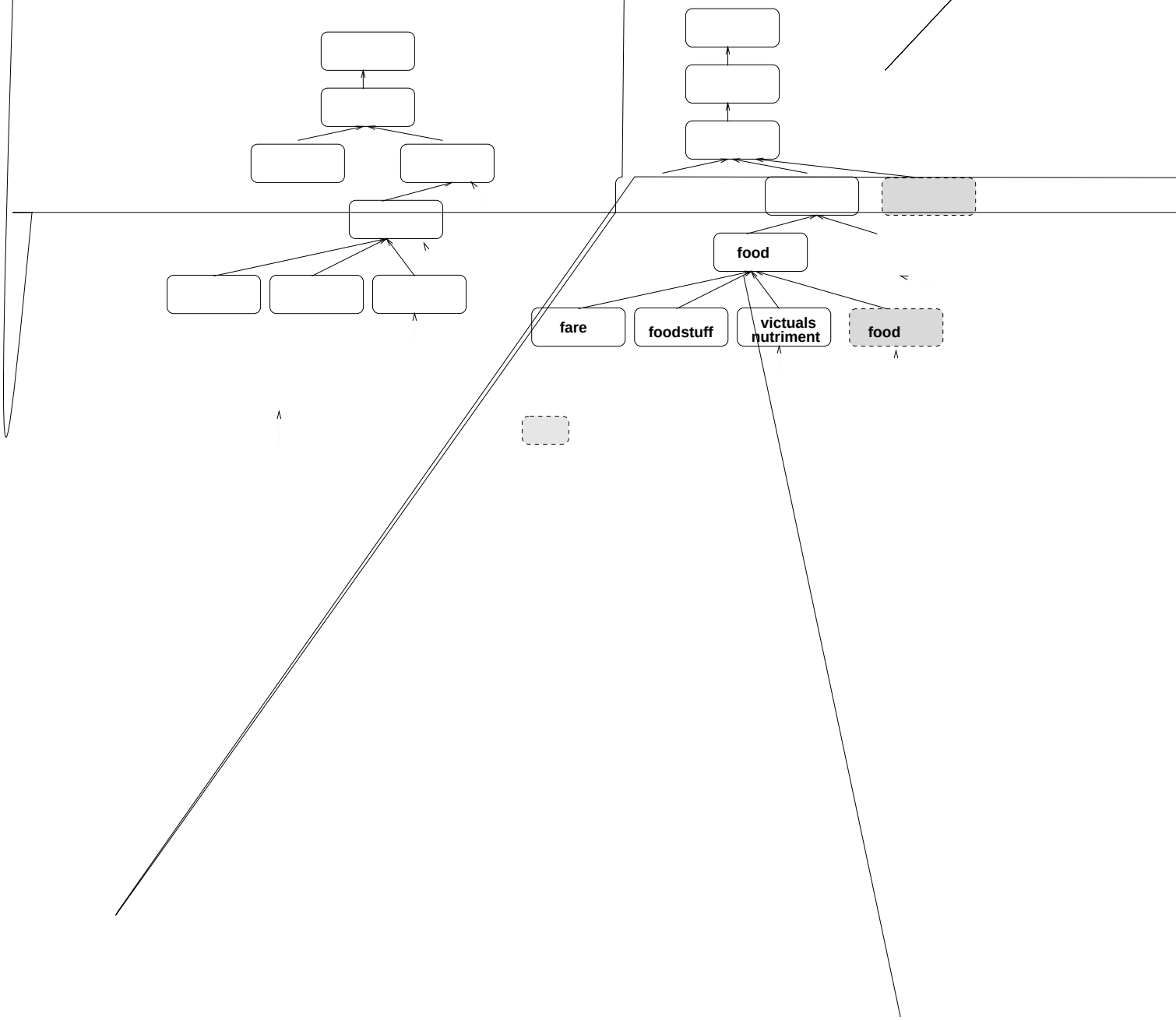


Figure 3: ATCM for ‘eat’ object slot.



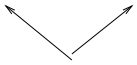
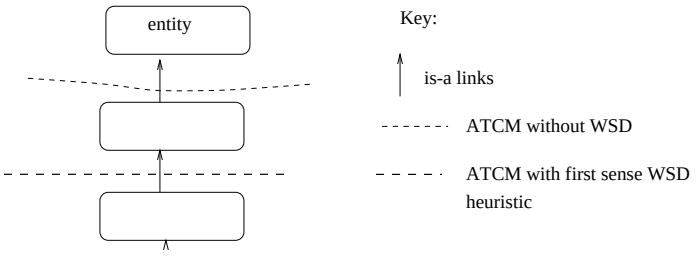


Table 7: CIDE Evaluation

Data (Million)	1st S	% correct	BL	diff
1.8	No	53	37	16
1.8	Yes	70	46	24
10.8	No	92	58	34
10.8	Yes	89	48	41



References

2 From internal reuse to external reuse

Software reuse is a general term that includes many different techniques. For example, consider the case of a programmer who needs to program something that he (or somebody else) has already programmed, or alternatively

there are cognitive differences between them.

In fact, what seems the most important at first is to find what precise knowledge should be displayed. Then, a precise study could be led to compare how various representations can display this type of information. The theories of Green et al. and von Mayrhauser and Vans could lead to such experiments.

3.3 Internalization

The internalization of programs is basically program comprehension, that is, the process of understanding a program using different sources, i.e. different external representations.

Soloway (Soloway & Ehrlich, 1984) argued that programming plans and discourse rules play a strong role in program comprehension. For example, he showed that expert programmers often make strong assumptions about programs, since they think that they all follow some common discourse rules.

By contrast, Pennington (Pennington, 1987) explains that programming plans just play a role in program comprehension and explanation, but that they are not the memory structure. The role of plans in internal representations of programs has not been, in her opinion, proven. As we see it, this confirms the distinction we made between general programming knowledge (which indeed seems to involve plans and rules) and internal representations of programs.

3.3.1 Models of program comprehension

Brooks (Brooks, 1983) described a model of program understanding based on hypothesis testing. To understand a program, the programmer has to make a global hypothesis on what the program does. Then he can split this hypothesis into sub-hypotheses, and so on, until the hypotheses attain a level of detail which make them comparable to the documentation available, e.g. code, texts, etc. A low-level hypothesis can thus be confirmed or disconfirmed by the documentation, in which case this hypothesis and its neighbours will have to be modified. This model is particularly focused on domain knowledge, or more precisely at how the program deals with the domain knowledge. Domain knowledge helps the reuser in formulating the hypotheses, and as a consequence, most hypotheses will be targeted at learning this domain knowledge.

Finally we can notice that what results from the comprehension process is an initial internal model of the program, which is stored in short term memory. It may well differ from the internal models, stored in long term memory, that a programmer has of previously used programs.

3.4 Externalization

Though not many studies have been undertaken on this subject, it may be interesting to look at how programmers externalize programs, that is how they express their knowledge of a program.

On one hand, this may tell us what programmers really know about programs. For example, having some programmers use a program, and then asking them to recall it at different moments in time could show us which program features they really remember. These features would be the basic knowledge held in the long-term memory internal representations. Such experiments have been led by Davies, but they were based at short-term memory. They confirmed the theory of focal lines in programming plans.

On the other hand, it might be interesting to see how programmers spontaneously express their knowledge (either programs, problems or requirements), and which external representations they naturally use. As we said before, Petre (Petre, 1990) found that, when programming, programmers used a “private, pseudo-language”. Other experiments of this type may point at which external representations may be the easiest to use for programmers.

The aim of all this would be to teach programmers what knowledge they should externalize in order to make their programs easily understandable by other people, and how.

4 Representations applied to reuse

The problem of studying the use of external memories and representations in software reuse is made more difficult by the fact that software reuse doesn’t just consist in looking at programs, be they internal or external. Each kind of software reuse is made of different successive steps, which have their own aims and requirements.

- *Solution evaluation*
  Finally, the programmer may evaluate how suitable the component is. Presumably, if it doesn't reach a min-

Besides, the programmer can describe two different things: a possible solution (which supposes that the programmer knows what he's looking for) or his problem (the tool then being able to match this problem to some solutions). Most keyword based tools allow the programmer to ignore this distinction, as keywords can indifferently describe the problem and the solution. By contrast, some natural language tools might provide a tool which will analyse the problem and find by itself what type of solution is required, though we still have to find how this can be done. The general idea is that it may well be simpler for a programmer, from a cognitive point of view, to express his problem rather than a possible solution, since problems are quite often already externalized, in the way of requirement drafts, contracts, etc.

As a conclusion, we can say that it may be interesting to conduct an experiment to support those ideas and see more thoroughly the advantages and shortcomings of the different methods of externalization.

- *Internalizing the component description*

  The other issue of searching in a database is to internalize the component descriptions, in order to find some components which, at first, might be suitable. Since thi

- *Intent* The short description of the purpose of this pattern
-

reuse system which would support such experiments. This system would be based on a set of modules, where each module can be chosen from a few alternative possibilities. These modules might include:

- the component type: code, design architecture, or design pattern
- what the user should externalize for the search: the problem or the behaviour
- how he should externalize it: using keywords, natural language, or by browsing through a tree of choices
- what knowledge should be displayed about each component for the early selection stage
- how this knowledge should be displayed: text, graphics, data or control flow diagrams, a combination of them, etc.
- what knowledge should be displayed, and how, for the cross-evaluation stage
- a specialization tool: to display further information, or to automate this process
- an integration tool

Lee, A., & Pennington, N. (1994). The effects of paradigm on cognitive activities in design. *The International Journal of Human-Computer Studies*, pp. 577–601.

McIlroy, M. D. (1968). Mass produced software components. In Naur, P., & Randell, B. (Eds.), *Software Engineer-*

In: Halloran, J. C. & Retkowsky, F. P. (Eds.). (1998). The 10th White House Papers: Graduate Research in the Cognitive & Computing Sciences at Sussex. Cognitive Science Research Paper 478, School of Cognitive & Computing Sciences, University of Sussex.

Intention movements and the evolution of animal signals

Jason Noble
jasonn@cogs.susx.ac.uk

School of Cognitive & Computing Sciences
University of Sussex
Brighton
BN1 9QH

Abstract Two animals contesting possession of a resource sometimes fight, but more often engage in aggressive displays until one or the other retreats. Ethologists such as Tinbergen and Lorenz suggested that such threat displays were honest signals of aggressive intent, and evolved through the ritualization of "intention movements"—movements that reliably predict subsequent behaviour. More recent thinking in theoretical biology has shifted to a gene-centred view where signalling is seen as the manipulation of one animal by another, but there is still room for the concept of intention movements as evolutionary seeds. A simulation, inspired by Maynard Smith's Hawk–Dove nar(m)10(o)19(e).008(e)20(e35(a)12

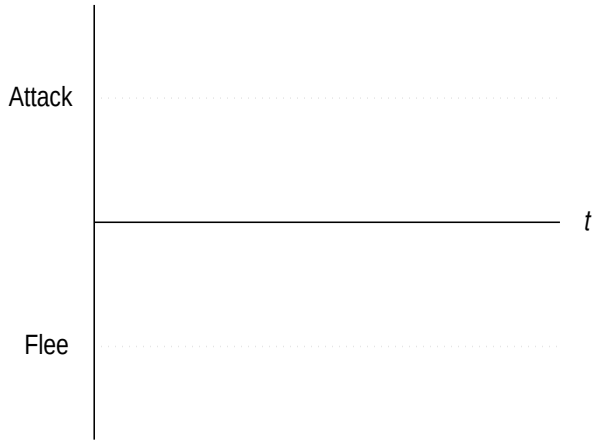
	. . . plays Hawk	. . . plays Dove
Hawk	-1	2
Dove	0	1

Table 8: Payoff matrix for the Hawk–Dove game

before biting, a mutant dog picks up on this and retreats when other dogs bare their teeth, and for a while the mutant does well because it avoids costly fights. However, once this "mind-reading" variant has become common in the population, the stage is now set for the arrival of a second mutant: a dog that bares its teeth even when it is weak or has no intention of fighting. The second mutant exploits the behaviour pattern of the first in order to ***manipulate*** animals that may be much stronger into walking away from the contest.

As you might guess, such a situation is not going to be stable. Dawkins and Krebs predict a cycle of increasingly inflated signals (e.g., teeth bared more prominently and for longer) and increasingly sceptical, "sales-resistant" receivers. The ethologists also thought that signals would be exaggerated, but in the interests of reducing ambiguity. In contrast, Dawkins and Krebs argue that it is rarely in an animal's interest to reduce ambiguity about its strength or its intention to fight.

Game-theoretic models have gone hand in hand with the gene-centred approach in biology. Game theory is a useful way of looking at problems in behaviour such as signalling, where the effectiveness of any one strategy depends on what other members of the population are doing. Maynard Smith (1982) developed the classic Hawk–Dove game (see table 8) to look at what happens when animals can either signal or fight over a resource. In the game, animals adopt one of two strategies: Hawk or Dove. Hawks always fight until they win or until they are seriously injured, whereas Doves try to settle the contest b


Attack
t
Flee

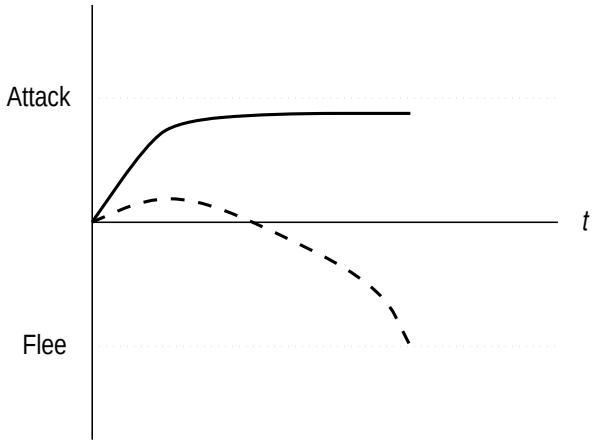


Figure 2: Winning a contest by threatening the opponent. The weaker animal (dashed line) abandons the resource and flees after a threat display by the stronger animal. Note that neither animal has actually attacked the other.

2.3 Behind the scenes

The animals were implemented as five-neuron fully inter-connected continuous-time recurrent neural nets, with all parameter values taken from Yamauchi and Beer (1994). This architecture was treated as a black box and I have assumed that it is capable of producing an approximation of any conceivable strategy; I have not tried to analyze the internal dynamics of the nets.

The genetic engine was a standard GA with a population size of 100, run for 5000 generations, using roulette-wheel selection. In each generation, animals were randomly selected to play out 500 contests; each animal could thus expect to play 10 games per generation.

2.4 Control groups

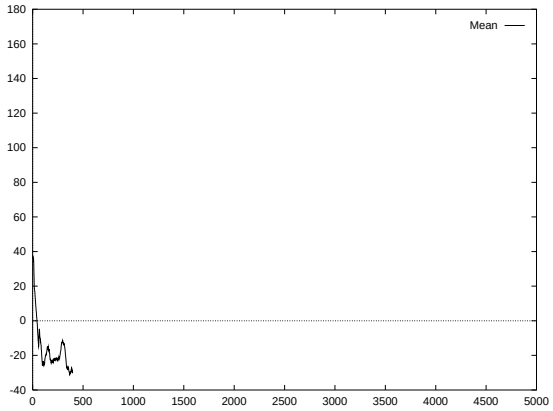
In order to tease out the patterns of causation, two control groups were devised.

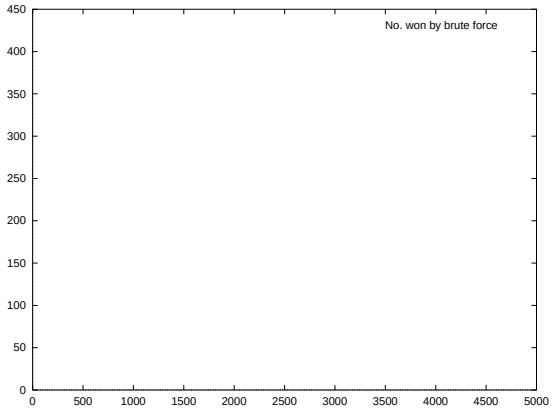
n **n** animals have no access to the position of their opponent on the Attack–Flee continuum. There is thus no possibility for communication and the animals must choose a strategy based only on their own strength.

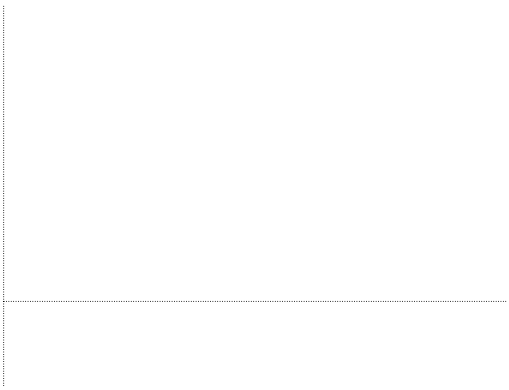
n **n** animals are permitted to know their opponent's strength. In a sense there will be no communication here either, because the main object of such communication has been served up for free. When animals are equally matched, perhaps they will pay attention to each other's intention movements, but when they are mismatched in strength they will know it immediately and presumably act accordingly.

3 Results

In simulation models like the one presented here, it is statistically desirable to present results that are averaged over a number of runs starting with different random seeds. However, the current model stands as a pilot study for future, more detailed investigations, and thus the results will refer only to a single, typical run in each experimental condition. The raw data, as is typical in these sort230.016(a)es


Mean
180
160
140
120
100
80
60
40
20
0
-20
-40
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000


No. won by brute force
450
400
350
300
250
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000



Grafen, A. (1990). Biological signals as handicaps. *Journal of Theoretical Biology*, *144*, 517–546.

Hurd, P. L. (1997). Is signalling of fighting ability costlier for weaker individuals?. *Journal of Theoretical Biology*, *184*, 83–88.

Krebs, J. R., & Dawkins, R. (1984). Animal signals: Mind reading and manipulation. In Krebs, J. R., & Davies, N. B. (Eds.),

In: Halloran, J. C. & Retkowsky, F. P. (Eds.). (1998). The 10th White House Papers: Graduate Research in the Cognitive & Computing Sciences at Sussex. Cognitive Science Research Paper 478, School of Cognitive & Computing Sciences, University of Sussex.

Teaching a Learning Companion

Jorge A. Ramírez Uresti *
jorgeru@cogs.susx.ac.uk

School of Cognitive & Computing Sciences
University of Sussex
Brighton
BN1 9QH

Abstract Learning Companion Systems (LCS) are a variation of Intelligent Tutoring Systems (ITS). In an LCS, besides the traditional tutor and student, a new agent is introduced: the Learning Companion (LC). The issues of the expertise and behaviour that such a companion agent should have, if it is going to be of any use to the student, are very important in these systems. This paper explores the hypothesis that a less capable learning companion is helpful to a human student by encouraging her to teach the LC.

Introduction

An Intelligent Tutoring System (ITS) can be seen as a system with two agents: a tutor and a student (figure 1). A criticism of such systems is that they are inherently based on one-to-one interactions between a student and a tutor and cannot encompass the richer learning possibilities opened up by involving more than one learner.

Learning Companion Systems (LCS) were first introduced by Ch

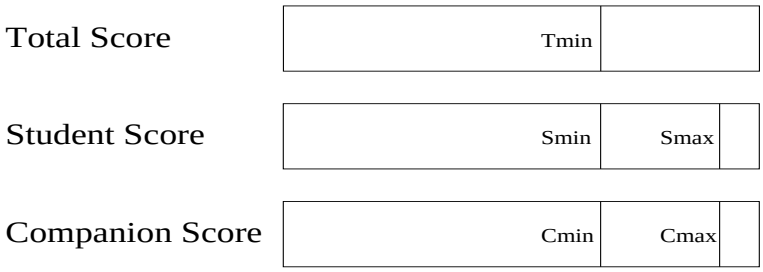


Figure 3: Motivation mechanism.

Chan, T.-W., & Baskin, A. B. (1990). Learning companion systems. In Frasson, C., & Gauthier, G. (Eds.), *Intelligent Tutoring Systems: At the Crossroads of Artificial Intelligence and Education*, chap. 1, pp. 6–33. Ablex Publishing Corporation, Norwood, New Jersey.

Cumming, G., & Self, J. (1989). Collaborative intelligent educational systems. In Bierman, D., Breuker, J., & Sandberg, J. (Eds.), *Artificial Intelligence and Education*, Vol. 3 of *Frontiers in Artificial Intelligence and Applications*, pp. 73–80 Amsterdam, Netherlands. IOS. Proceedings of the 4th International Conference on AI and Education.

Gilmore, D., & Self, J. (1988). The aplication of machine learning to intelligent tutoring systems. In Self, J. (Ed.), *Artificial Intelligence and Human Learning, Intelligent Computer-Aided Instruction*, chap. 11, pp. 179–196. Chapman and Hall Computing.

Goodlad, S., & Hirst, B. (1989). *Peer Tutoring: A Guide to Learning by Teaching*. Kogan Page, London.

In: Halloran, J. C. & Retkowsky, F. P. (Eds.). (1998). The 10th White House Papers: Graduate Research in the Cognitive & Computing Sciences at Sussex. Cognitive Science Research Paper 478, School of Cognitive & Computing Sciences, University of Sussex.

Debugging as program comprehension

Pablo Romero Mares
juanr@cogs.susx.ac.uk

- Turn on the trace.
- Issue the goal.
- *creep* to examine the subgoals.
- *skip* over each subgoal.
- If an incorrect result is detected, *retry* the last subgoal.
- *creep* to examine the behaviour of the defective subgoal's subgoals.
- Repeat the process for the new set of goals.

Figure 1: General strategy to go from the ***symptom*** to the ***program misbehaviour*** description
From Brna et al. (1992)

It can be noted that this strategy is based on a top-down methodology for the debugging task. This top-down methodology assumes that programmers will try to find errors by querying the main predicate of the program and will advance querying subgoals until they find the problematic procedure. This strategy assumes that programmers will mainly apply a data gathering technique when debugging a program.

Brna et al. give a detailed description of the strategy to debug non-terminating programs. For the purpose of comparing the debugging behaviour of subjects in the experiment reported in this paper with the debugging strategy for non-terminating programs, the end of this section talks about the sub-case of Malignant Endless Building.

Brna et al. subdivide the non-terminating program problem into different categories. One of these has to do with the case in which, through a step trace of the program, subgoals for the query under current investigation

behaviour10(n)-20 of s0146(b)-4(j)-10(e)12(c)12(i)-9.99ss that p-10(e)12(r017(f)-3(o)-4(e)12(m)10.0-15(e)11.9)12(n)-195.985(

```
component(earpiece, e32).
component(mouthpiece, p24).
component(cord, sc27).
component(body, bb49).

unit(handset, [earpiece, mouthpiece]).
unit(phone, [handset, body, cord]).

list_components(Component, List) :-
        component(Component, List).

list_components(Component, FList) :-
        unit(Component, CList),
        process(CList, Flist).

process([],X).

process([Component|Tail], Full_list) :-
        list_components(Component, Comp1),
        prcess(Tail, Rest),
        append(Comp1, Rest, Full_list).

append([X|Xs], Ys, [X|Zs]) :-
        append(Xs, Ys, Zs).

append([],Zs,Zs).
```

Figure 2: Buggy code for the Phone Components program

(From Mulholland (1995))

4.3 Materials

The program to be debugged was a small 19 line Prolog program that performed a simple database retrieval of

Errors / subjects	S1	S2	S3	S4	S5	S6
*list_components/2*a	f	c	c	c	c	
*list_components/2*b			c	c		c
*process/2*a			c	c	c	
*process/2*b	c	c	c	c	c	c

- (f) found
-

example, having a variable that was never instantiated, or a base case for processing a list that did not consider the empty list as one of its arguments).

Data gathering was, as mentioned above, one of the most frequently used debugging techniques. This technique was performed through the application of queries to the Prolog interpreter. Subjects used queries to narrow the bug's possible location. Asking queries to the interpreter was an easy way to find out if a procedure was working correctly, or if it had some apparent problem. Therefore, programmers could ignore correct procedures and concentrate on those that had an undesired behaviour. Asking queries to the interpreter was also used to test hypothesis about the program behavior or to test the correctness of a modification to the program. Figure 3 shows part of the interpreter history annotated

```
‘well I’ll first test one of the queries of the handout, the
first. Just to see what happens.’

?- process([earpiece,mouthpiece],L).
no

‘Process says no. Ok, so I see process looks like is a recursive
procedure. Oops, I see a typo in process second clause, so instead of
process you’ve got prcess, so let’s change that, load that again, and
test it again.’

?- process([earpiece,mouthpiece],L).
no

‘no, so also it seems that append is defined with the base case at the
bottom. Append again is recursive, and it looks like’

?- append([a,b,c],[d,e,f],L).
L = [a, b, c, d, e, f] ?
yes

‘right append looks ok. So may be list_components, ok, so
list_components tries to find out whether the first component is just
a component or whether it can be broken down, for example handset
which is earpiece and mouth piece. Test list_components, first of all
just with earpiece,’

?- list_components(earpiece, L).
L = e32 ?
yes

‘and now with the handset.’

?- list_components(handset,L).
no

‘Aha!, so list_components fails with a handset, so its the second
clause with handset. That does a recursive call to process... This
stopping condition is slightly worrying for process,  because given
an empty list I guess it should return an empty list, but is
undefined. So, lets change this condition... and I’ll try
list_components again.’

?- list_components(handset,L).
no

‘No, right. Ok, so lets test the stopping condition...’

?- process([],L).
L = [] ?
yes
```

Figure 3: Annotated interpreter history for subject S3

```
‘...I don’t understand the process procedure base case. I’ll try with
an spy in list_components and a simple query’

?- process([earpiece],L).
** (1) Call : process([earpiece], _1)?
** (2) Call : list_components(earpiece, _2)?
** (2) Exit : list_components(earpiece, e32)
** (3) Call : process([], _3)?
** (3) Exit : process([], _3)?
** (3) Redo : process([], _3)?
** (3) Fail : process([], _3)?
** (2) Redo : list_components(earpiece, e32)?
** (2) Fail : list_components(earpiece, _2)?
** (1) Fail : process([earpiece], _1)?
no

‘list_components, and should enter here. Goes out of the base case
and it looks ok. An then, it fails, here, when it redt
```

‘So the first thing I’ll do is just see what works. Let’s try something else. Ok, it’s gonna give no for everything so I’ll look at the program. Process, two clauses, process list, the empty list will get at, uhm. So let’s try that [process/2]. Process, empty list, L. Oh, so process component, tail, so there is a mistake here [process/2b error] so that’s something. May be I’ll check for more mistakes... Something I half remember, I’m not sure if it is right or not. I’m sure the stopping clause has to come first, fairly sure...’

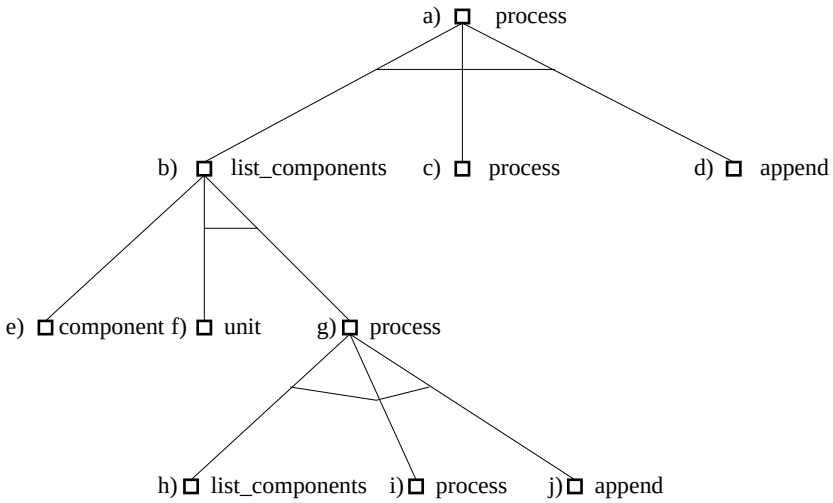
Figure 5: Fragment of subject S6 comments

4.5.2 Differences between successful and unsuccessful subjects

One important point when considering the explicit teaching of debugging is to observe the differences in debugging behaviour between successful and unsuccessful subjects. Apparently, successful and unsuccessful subjects applied similar techniques. However, there were differences in aspects such as: which techniques were most common among each of these subgroups, the results of applying specific techniques, the quality of the structural knowledge they have about the language, the interpreter tools and the notional machine, how global or local the comprehension process was and which parts of the task they focused on.

There were some techniques that were applied more frequently among the successful subjects. These techniques are *visual inspection of code* and *checking coding conventions*. In general, successful subjects would tend to apply visual inspection of code episodes frequently, mostly at early stages of the debugging session. Subjects would use these episodes to form an early, rough idea of what the code was doing. Successful subjects were able to put into words this early mental model they formed. Most of the time their claims represented a good approximation to the code structure and behaviour. These subjects would also apply checking coding conventions episodes to spot errors before coming across their symptoms. It seems that their strong knowledge of the language syntax and coding conventions allowed them to apply this technique. Less successful subjects would also spot apparently suspicious segments of code, but

‘Ok, the


a) process
b) list_components
c) process
d) append
e) component
f) unit
g) process
h) list_components
i) process
j) append

description to the *Program code error* description might not be straightforward. In fact, if the gap between the place the error is generated and the place the symptom of the error appears, trying to obtain a complete *program misbehaviour* description might be misleading.

The debugging session of subjects S3 and S4 gave a chance to analyse the case of non-terminating programs. These two subjects corrected the *process/2*a, *process/2*b and *list_components/2*a errors and then tried a query to test the main procedure for the case of the handset. This resulted in a non-terminating call to the *append/3* procedure. This non-terminating call was due to the fact that *append/3* was called with an uninstantiated variable as first argument because the *list_components/2* second predicate was losing a variable (the *list_components/2*b misspelling error). The uninstantiated variable that

debugging session makes it difficult attempting to characterise each one of these activities separately. One solution to this problem could be to design experiments w

Kessler, C. M., & Anderson, J. R. (1986). A model of novice debugging in lisp. In *Empirical Studies of programmers, first workshop* Norwood,NJ. Ablex.

Mulholland, P. (1995). *A Framework for Describing and Evaluating Software Visaualization Systems: a Case Study in Prolog*. Ph.D. thesis, Knowledge Media Institute, Open University, Milton Keynes, U.K.

follicle–stimulating hormone (FSH), luteinizing hormone (LH) and luteotropic hormone (LTH). Based on endocrine events, the menstrual cycle can be divided into 3 phases: follicular, ovulatory and luteal phases. Figure 1 shows the changes in hormone levels during the menstrual cycle (Merck, 1997).

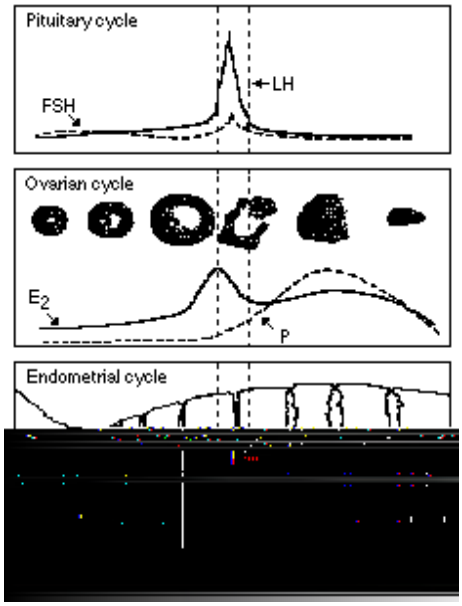


Figure 1: Hormonal changes in menstrual cycle

Follicular and luteal phases of the menstrual cycle are each 14 days long. During the follicular phase, the ovaries produce increasing amounts of oestrogen. In the first half of this phase, secretion of FSH slightly increases and causes the ovarian follicles to grow and develop. In the second half, secretion of oestrogen, particularly oestradiol (E_2), from the ovaries increases slowly and then accelerates and peaks during the day of LH surge. Just before the LH surge, progesterone levels also begin to increase. During the early luteal phase the *corpus luteum* supports the released ovum by secreting increasing quantities of progesterone and oestrogens, while LH and FSH levels drop down. During the late luteal phase, circulating levels of LH and FSH remain low, but begin to increase again with menstruation. Oestrogen and progesterone levels tend to decline in this part of the cycle. The ovulatory phase is 1–2 days long. During this time, a large amount of LH, called the preovulatory surge, is secreted by the pituitary gland and is necessary for the final follicular growth and ovulation.

Cyclical changes in the breast have been identified by invasive and non–invasive means. Both approaches have observed changes due to hormone influence. These changes are more likely to occur in the epithelium, rather than in stromal tissues, because epithelial cells are more sensitive to changes in oestrogen and progesterone levels (Payne et al., 1994; Longacre & Bartow, 1986; von Schoultz, Söderqvist, von Schoultz, & Skoog, 1996; Fanger & Ree, 1974; Potten, Watson, Williams, Tickle, Roberts, Harris, & Howell, 1988; Ferguson & Anderson, 1981; Twelves et al., 1994). Although these metabolic variations can be monitored with ^{31}P MRS, the complexity of the information prevents the expert from unveiling those subtle features embedded in a spectrum that can lead into the identification of certain patterns of cell behaviour.

It has been confirmed by several authors that in pre-menopausal women, maximal epithelial cell proliferation occurs during luteal phase of menstrual cycle (Strnad, Vachoušek, Čech, Smejkal, & Velek, 1995; Nazario, Simoes, & De Lima, 1994; Nazario, De Lima, Simoes, & Novo, 1995; von Schoultz

If

```
rule(1, ( (ef, ?x) <- (prolif_rate, ?x))).
rule(2, ( (lf, ?x) <- (prolif_rate, ?x))).
rule(3, ( (ef, ?x) <- (pi_level, value))).
rule(4, ( (el, ?x) <- (metab_act, ?x))).
rule(5, ( (ll, ?x) <- (metab_act, ?x))).
rule(6, ( (el, ?x) <- (pme_level, value))).
rule(7, ( (ll, ?x) <- (pme_level, value))).
rule(8, ( (el, ?x) <- (pi_level, value))).
rule(9, ( (ll, ?x) <- (pi_level, value))).
rule(10, ( (ll, ?x) <- (prolif_rate, ?x))).
rule(11, ( (prolif_rate, ?x) <- (pme_level, value))).
rule(12, ( (lf, ?x) <- (pde_level, value))).
rule(13, ( (bio_changes, ?x) <- (pde_level, value) &
                                (pme_level, value))).
rule(14, ( (el, ?x) <- (bio_changes, ?x))).
rule(15, ( (ef, ?x) <- (bio_changes, ?x))).
rule(16, ( (ll, ?x) <- (bio_changes, ?x))).
rule(17, ( (lf, ?x) <- (bio_changes, ?x) &
                       (pme_level, value))).
```

16 *in vivo* ^{31}P spectra from four normal breast from female pre–menopausal volunteers, four spectra from each volunteer, one for each phase of the menstrual cycle, were used for training the networks. In a preliminary stage, all the 16 cases were used as training set to evaluate the performance of knowledge-based networks with different topologies, as well as knowledge-free networks. *Leave-one-out* method, as it says, leaves one sample out of the training set. A network is trained with $n-1$ cases and then is tested with the remaining case. This process is repeated until the network has been trained with n possible sets. Its performance is the overall average of n trials.

7 Results

31 knowledge-free networks, with one to 20 hidden units were trained with different learning rate, momentum and initial random weight values. A very poor performance was observed in all of them, with an average pattern/error of 0.7500 and standard deviation of $1.2615x10^{-4}$. For the knowledge-based networks, 5 different topologies were created from 5 different sets of rules extracted from the main knowledge base mentioned above. All of them were trained with different learning rate and momentum values. Random noise was added for perturbing the initial weights and biases. Thus, there was a total of 42 networks for each topology. Both knowledge–free and knowledge–based networks had 7 inputs corresponding to the area under each peak of metabolites from *in vivo* ^{31}P spectra from normal breast tissues: PME, PDE, PCr, Pi, α–ATP, β–ATP and γ–ATP. They also had four outputs corresponding to each phase of the menstrual cycle: early follicular (EF), late follicular (LF), early luteal (EL) and late luteal (LL).

Average pattern/error and standard deviation, two measures to evaluate the performance of a network and set of networks with the same characteristics (e.g. topology, learning rate, momentum, initial random added noise), were used. The average pattern/error for a single network after training or testing is given by equation 8.

$$avg.\ pattern/error = \frac{total\ error}{number\ of\ patterns} \tag{8}$$

Similarly, the average pattern/error for a batch of networks is given by 9, where $avg.\ pattern/error_i$ is the average pattern/error for the network$_i$ given by eq. 8 and n is the total number of networks.

$$avg.\ pattern/error = \frac{\sum_{i=1}^{n} avg.\ pattern/error_i}{n} \tag{9}$$

The standard deviation (std) for a set of networks is given by equation 10, where $error_i$ is the average pattern/error of network$_i$, $avg.error$ is the total average pattern/error given by eq. 9, and n is the total number of networks in the batch.

$$std = \sqrt{\frac{\sum_{i=1}^{n} (error_i - avg.\ error)^2}{n-1}} \tag{10}$$

Table 11 shows how performance improves as the number of rules included in the knowledge base increases. A set of KBANN's with a topology defined from the whole set of 17 rules (section 4) had 0.1779 average pattern/error and 0.0269 standard deviation, whereas networks with knowledge bases of 11 rules (No. 3, 6–9, 12–17), 9 rules (No. 3, 8, 9, 12–17), 6 rules (No. 12–17) and 5 rules (No. 13–17) had average pattern/ errors of 0.2731, 0.2829, 0.3134 and 0.3662, and standard deviations of 0.0271, 0.0342, 0.0465 and 0.0286 respectively. These results, when compared to those from knowledge–free

Testing Phase			
	Performance		
Network	average pat/error	std	Correct (%)
KBANN-10	0.4380	0.3617	8/16 (50%)
KBANN-12	0.6280	0.3684	4/16 (25%)
KBANN-13	0.4729	0.3548	7/16 (44%)

Table 13: KBANN: *leave–one–out* method

8 Discussion

Towell, G. G. (1991). *Symbolic Knowledge and Neural Networks: Insertion, Refinement and Extraction*. Ph.D. thesis, University of Wisconsin, Madison.

Twelves, C., Porter, D., Lowry, M., Dobbs, N., & *et.al.* (1994). Phosphorus–31 metabolism of post–menopausal breast cancer studied *in vivo* by magnetic resonance spectroscopy. *British Journal of Cancer*, *69*, 1151–1156.

Vogel, P., Georgiade, N., Fetter, B., Vogel, F., & McCarty, K. (1981). The Correlation of Histologic