Volume 18 Number 4 December 1994 ISSN 0350-5596

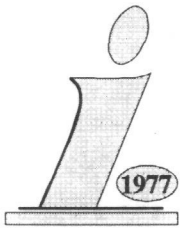
Informatica

An International Journal of Computing and Informatics

Special Issue: Artificial Life

Guest Editor: Xin Yao

Informatica 18 (1994) Number 4, pp. 375–507



The Slovene Society Informatika, Ljubljana, Slovenia

Informatica

An International Journal of Computing and Informatics

Basic info about Informatica and back issues may be FTP'd from ftp.arnes.si in magazines/informatica ID: anonymous PASSWORD: <your mail address>
FTP archive may be also accessed with WWW (worldwide web) clients with URL: ftp://ftp.arnes.si/magazines/informatica

Subscription Information: Informatica (ISSN 0350-5596) is published four times a year in Spring, Summer, Autumn, and Winter (4 issues per year) by the Slovene Society Informatika, Vožarski pot 12, 61000 Ljubljana, Slovenia.

The subscription rate for 1994 (Volume 18) is
- DEM 50 (US$ 34) for institutions,
- DEM 25 (US$ 17) for individuals, and
- DEM 10 (US$ 4) for students
plus the mail charge DEM 10 (US$ 4).

Claims for missing issues will be honored free of charge within six months after the publication date of the issue.

LaTeX Tech. Support: Borut Žnidar, DALCOM d.o.o., Stegne 27, 61000 Ljubljana, Slovenia.
Lectorship: Fergus F. Smith, AMIDAS d.o.o., Cankarjevo nabrežje 11, Ljubljana, Slovenia.
Printed by Biro M, d.o.o., Žibertova 1, 61000 Ljubljana, Slovenia.

Orders for subscription may be placed by telephone or fax using any major credit card. Please call Mr. R. Murn, Department for Computer Science, Jožef Stefan Institute: Tel (+386) 61 1259 199, Fax (+386) 61 219 385, or use the bank account number 900–27620–5159/4 Ljubljanska banka d.d. Slovenia (LB 50101-678-51841 for domestic subscribers only).

According to the opinion of the Ministry for Informing (number 23/216-92 of March 27, 1992), the scientific journal Informatica is a product of informative matter (point 13 of the tariff number 3), for which the tax of traffic amounts to 5%.

Informatica is published in cooperation with the following societies (and contact persons):
Robotics Society of Slovenia (Jadran Lenarčič)
Slovene Society for Pattern Recognition (Franjo Pernuš)
Slovenian Artificial Intelligence Society (Matjaž Gams)
Slovenian Society of Mathematicians, Physicists and Astronomers (Bojan Mohar)
Automatic Control Society of Slovenia (Borut Zupančič)
Slovenian Association of Technical and Natural Sciences (Janez Peklenik)

The issuing of the Informatica journal is financially supported by the Ministry for Science and Technology, Slovenska 50, 61000 Ljubljana, Slovenia.

The Artificial Evolution of Adaptive Processes

Douglas T. Crosher
School of Electrical Engineering
Swinburne University of Technology
P.O. Box 218, Hawthorn 3122, Australia
E-mail: dtc@scrooge.ee.swin.oz.au

Keywords: evolution, learning, adaptation

Edited by: Xin Yao
Received: November 16, 1993 **Revised:** April 11, 1994 **Accepted:** April 18, 1994

It is hypothesised that the answer to the question of how adaptive or learning processes can evolve is through an appropriately designed evolutionary search domain, search technique, and problem environment. A representation is described that is able to represent a general class of adaptive processes. The hypothesis is explored through three experiments with a fixed evolutionary search algorithm and graded problems. The search algorithm is able to find solutions to two of the problems but fails on a slightly more difficult problem. The failure is explained by the lack of a priori knowledge, thus supporting the hypothesis. The implications for the study of the evolution of learning are discussed.

1 Introduction

The general question that this paper contributes to is: how do adaptive processes evolve? The hypothesised answer to this question, with the assumptions mentioned below, is that adaptive processes will emerge through an evolutionary search if the designer has chosen *a priori* an appropriate evolutionary search domain, search technique, and an appropriate environmental task requiring learning.

The following assumptions are made about the model of evolution:

- Assuming a fixed population size with each generation being marked by the testing and assignment of a resultant score to each member of the population.
- Assuming a statistically stationary environment in which each member of the population is independently tested (The members do not interact during their life span).
- Assuming that at the end of each generation a new generation is created from the previous generation through both selection based on the score of the members, and the application of random changes to randomly chosen members.

These assumptions are made in the studies of a number of researchers who address the evolution of learning. Typically additional specific assumptions need to be made to achieve success. Miller et. al. (Miller et. al. 1989) describe the design of neural networks using genetic algorithms, they assume a feedforward neural network and use a genetic algorithm to search for a constraint matrix that defines the connections between these neurons. David Chalmers's (Chalmers 1990) studies the evolution of the updating function used by neural networks, which was followed up by Fontanari and Meir (Fontanari & Meir 1991) evolving a learning algorithm for the binary perceptron.

The search domain is largely determined by the representation of possible adaptive algorithms, and partly by the evolutionary search algorithm which is an evolutionary strategy. An original representation was iteratively developed such that it had enough generality to represent a class of adaptive and learning algorithms, while at the same time being specific enough that the search algorithm would be able to find at least minimal

solutions to some problems. This representation describes the adaptive mechanism at a more general level than considered by Todd and Miller and Chalmers, it consists of memory locations connected via addition and multiplication operations.

To add support to the hypothesis a series of three experiments was conducted each with a different problem, while obtaining two successes and a failure. Finally it is shown that the failure can be trivially overcome by suitably biasing the search algorithm.

The problems are graded in difficulty with respect to the search algorithm. The first is borrowed from control theory, the task is to control a plant to track a reference signal, the plant being an integrator. The second problem is the habituation and sensitisation task described by Todd and Miller (Todd & Miller 1990). The third problem is the associative learning task described by Todd and Miller (Todd & Miller 1991) for which the evolutionary search fails to find a solution even after lengthy simulation runs. However by biasing the search algorithm, through an initial solution, a solution to the third problem was trivially found.

Below the three experiments are presented each with a minor discussion, followed by a concluding discussion of the implications of the hypothesis, its relation to relevant research programs, and suggestions for further lines of study. But firstly the search algorithm is described as it is common to all the experiments.

2 The search algorithm / evolutionary strategy

This section describes the evolutionary search algorithm, this can be broken down into the representation of the functionality of each member of the population, and the evolutionary model including its initialisation, its main loop, and the termination.

2.1 The representation / functionality

The goal in the design of the representation was to be able to represent the functionality of a class of adaptive algorithms, and to bias the search so that solutions could be found to the planned problems.

It was decided to explore a class of adaptive algorithms that could represent solutions to linear and non-linear discrete control problems, and solutions to some simple unsupervised learning problems. The functionality of this class was achieved through networks of memory nodes connected via addition and multiplication functions.

To facilitate the smooth evolution of solutions two representational features were decided upon. Firstly a parallel executional model was used, and secondly at each time step the value of a node was updated with the *sum* of its inputs.

The choice of the parallel computational representation had two advantages over a sequential computational representation.

Firstly in a parallel model the functionality is a lot more independent than in a sequential model which aids a smooth incremental evolution through random changes. For example the mutation of one instruction in a sequential model can effect the course of all other instructions, whereas in a parallel model the effect of a random mutation is likely to have little effect on the functioning of the larger system.

Secondly it was considered desirable to have the evolutionary search freely explore the performance of large networks. With a parallel representation the time delay can be independent of the number of nodes and functions. Whereas with a sequential representation time delays can grow as the number of steps in its evaluation loop increases, this can decrease performance and bias the evolutionary search away from exploring larger networks.

The decision to have functions summing into the nodes was based on the general desirability of linearity within search domains, and also to support the smooth addition of functionality to a network through the evolutionary search.

The model settled on is a network of nodes. The nodes have a combination of two functions summing into them, these functions are the sum of constants, and the sum of the product of pairs of nodes. At each time step the nodes are updated, the new value for a node n_i is calculated as follows:

$$n_{i_{t+1}} = \sum k_j + \sum n_{x_t} n_{y_t}$$

This model can be represented graphically in the same way that a discrete signal processing or control algorithm might be, see Figure 1.

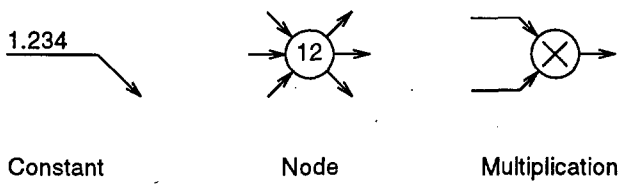


Figure 1: Graph symbols

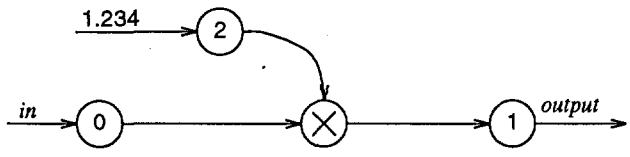


Figure 2: Example population member

The interaction of a member of the population with its environment was handled by having special nodes for its inputs and outputs. The input nodes were clamped to signals from the environment, and the output nodes were passed back to the environment.

An example is shown in Figure 2. It consists of three nodes, one input, one output, and one normal node. Two functions are show, there is one constant summing into node 2, and the product of nodes 2 and 0 sum into node 1. Mathematically the network would behave as follows:

$$\begin{aligned} n_{1_{t+1}} &= n_{2_t} n_{0_t} \\ n_{2_{t+1}} &= 1.234 \end{aligned}$$

Since node 0 is clamped to the input and the output is taken from node 1, and noting that node 2 is constant, the above equations are equivalent to:

$$output_{t+1} = 1.234 \times in_t$$

2.2 The evolutionary search model

An Evolutionary Strategy (ES) was chosen for the simplicity of avoiding a crossover operator. This is in contrast to the more commonly used Genetic Algorithms (GA) in which the genetic crossover operator is employed (Goldberg 1989). The use of an ES frees up restrictions that would be present with a GA. There is a growing awareness that GAs do have limitations (Forrest & Mitchell 1993, Forrest & Mitchell 1992), and of the abilities of ESs.

The procedure can be broken down into the initialisation, a main loop, the termination.

2.2.1 Initialisation

The initialisation was simple, consisting of setting up a population in which each member had only a minimal number of nodes for its communication with the environment. Needless to say the initial performance is poor! The population size used in the experiments was 100 members, this was not a critical factor, typically an increase in the population size reduced the number of iterations to find a solution.

2.2.2 The main loop

Testing consists of evaluating the functional network of each member of the population in the test environment. Three environments are described later, at this point it is sufficient to know that as a result of the interaction of the members networks with the environment they are assigned a performance result. So after the testing stage each member has a score which is a function of how well it performed.

After all the members have been tested a new generation is formed by selecting members from the previous generation with a probability proportional to their performance, and by mutating some of them.

The way in which the search algorithm mutated the existing members of the population was critical to it finding a solution as this is a major contributor to the exploration bias of the algorithm. Three classes of changes could be made, network additions, constant mutations, and network deletions.

The expected number of network additions to a member was 33% of the current network size. It needed to be scaled by the current network size because of the diminishing effect of a function addition as the network grows large. Each member had a limit to the size of its network which was 100 nodes and 100 constant and multiplication functions. If a member had free space then a network addition could occur. There were three types of network additions, the addition of a con-

stant summing into a node, the addition of a multiplication function summing into a node with its input determined randomly, and finally the addition of a multiplication function summing into a node with one input taken from a node and the other from a newly created constant node.

There was 10% chance of a constant being scaled by a random factor between 0.0 and 2.0.

There was a 33% chance that a constant or multiplication network function would be removed. This helped to clear out junk nodes that made no contribution and to free up space for the addition of new nodes.

2.2.3 Termination

The main loop would cycle and the user was able to watch the simulation proceeding, after it had found a stable solution the user could stop it and examine the result. However as the simulation proceeds the network size grows until it reaches the limit. Trying to interpret a member was almost impossible due to the large number of *junk* nodes and functions which had little or no effect. Often it was difficult to identify what was junk.

To overcome this difficulty and to facilitate an analysis the user was allowed to trigger the program to stop adding new instructions near the end of the experiment. Without the continual addition of new instructions and with continuing instruction removals all the junk instructions are removed and you are typically left with less than 10 instructions to scrutinise, and you can be assured that they are all necessary to maintain performance.

3 Experiment 1: Evolution of a feedback controller

This is the first of the experiments, the evolutionary search algorithm described above is applied to evolve solutions of the following problem. The members of the population are controllers which have two inputs, a reference which it is required to track and the plant output. It also had one output which was the input to the plant. See Figure 3. On each iteration a member's node 0 is clamped to the reference input, node 1 is clamped to the plant output, and the controller output is read from node 2.

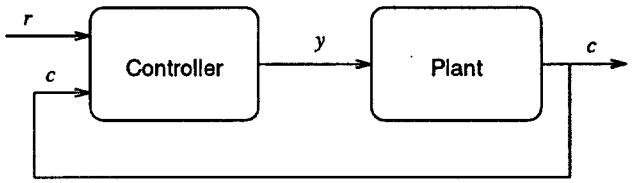


Figure 3: Control problem

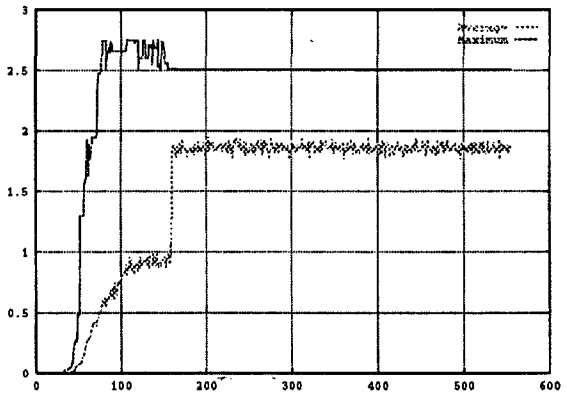


Figure 4: Evolution of feedback controller

The reference signal was always the same, the equation used was:

$$r = 100.0 \times \sin(\frac{2 \times 3.1415 \times t}{5000.0})$$

The plant used is a simple integrator which is not an uncommon problem. The update equation for the plant output is:

$$c = c + \frac{y}{100.0}$$

The score s of a member was:

$$s = \frac{1}{\sqrt{\Sigma (r-c)^2}}$$

3.1 Results

Figure 4 shows how the search proceeded. It shows the population maximum and average plotted against the number of generations.

A detailed analysis of the resultant evolved network of one characteristic and successful member (score on 2.505101) from the run is presented below, including a graph of its network in Figure 5,

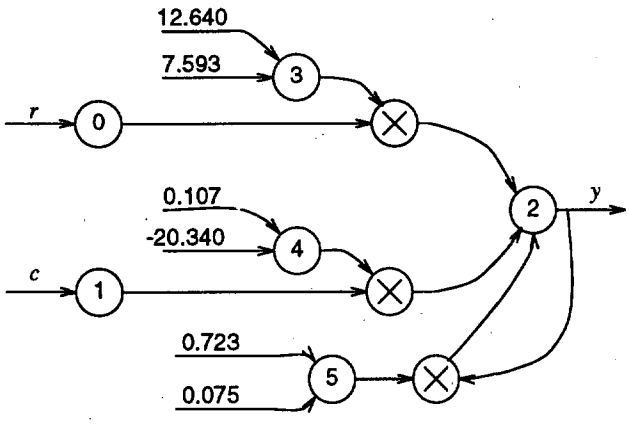


Figure 5: Graphical representation of member 0

and a mathematical simplification of the network below.

The graph can be simplified as follows:

$$\begin{aligned} y_{t+1} &= (12.640 + 7.593)\, r_t \\ &\quad + (-20.340 + 0.107)\, c_t) \\ &\quad + 0.798 \times y_t \\ &= 20.233 \times r_t - 20.233 \times c_t + 0.798 \times y_t \end{aligned}$$

Now it can be noticed that the factors of r_t and c_t both come to 20.233, this close correlation is explained with the following step in which is is shown that it is solving the control problem by acting as a negative feedback controller. So assuming the two factors are equal allows the rewriting of the equation in the form:

$$y_{t+1} = \underbrace{20.233\,(r_t - c_t)}_{\text{Negative feedback}} + \underbrace{0.798 \times y_t}_{\text{Filter}}$$

Now the negative feedback can be clearly seen. The two factors are held at the same value because any drifting apart would degrade the performance of the controller, and it would be selected against by the evolutionary algorithm. The requirement for coordination in the changing of these factors makes it unlikely that the evolutionary algorithm could adjust the gain if required. It has become trapped in a local optimum.

Although it is easy to imagine a configuration in which the gain could be in a single parameter, the search consistently discovers the above configuration. In any case there is no selective pressure

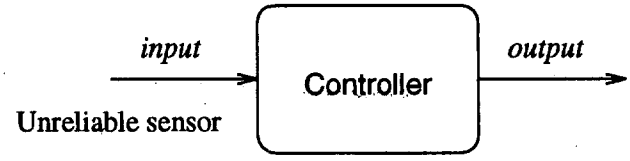


Figure 6: Evolution of sensitisation and habituation

to choose this configuration over the other, they both solve the problem perfectly. Only if the problem were changed to require rapid gain changes would there be any benefit in choosing one over another. This draws out a weakness in this model of the evolution of adaptability, with a fixed problem there is a definite limit to what is meant by adaptability.

Typically also simple feedback loops also evolved although there was variation between runs. These feedback loops were unexpected. In the case above the loop acts to filter the output which gave some benefit.

4 Experiment 2: Evolution of sensitisation and habituation

The next problem to be studied is based on the work of Todd and Miller (Todd & Miller 1990). They created an environment and an evolutionary ANN in which habituation and sensitisation emerge. In an environment in which good and bad events do not occur at random but are clumped together, and in which there is sensor noise, it is found that 'cluster-tracking' is of adaptive value. The problem is illustrated in Figure 6.

Each member has one input and one output. It is simulated in an environment for $1000 * 2$ time steps. The input to a member was an unreliable sensor of a slowly varying signal. The member received an increase in its score if it acted when the signal was at a high level, but due to the unreliable sensor members may mis-interpret the signal which could lead to incorrect decisions.

Based on the results of Todd and Miller, in order to encourage the evolution of a solution, the sensor was given a 75% chance of being correct and a 25% chance of being incorrect. To allow for propagation through the network the input was set for two time steps between each update, and

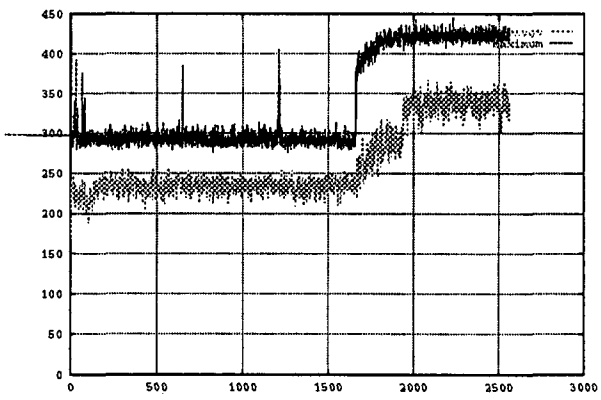


Figure 7: Evolution of sensitisation and habituation

was converted to an analog value of positive or negative one.

The analog output value was converted to a digital value via a threshold function (if it was greater than one then it indicated an output action). If an output was indicated then the score was increased if the true environmental signal was high else the score was decreased.

The environmental signal was slowly changing from one state to another every $50 * 2$ time steps. A member which evolves an ability to smooth the signal can extract a cleaner measure of the real signal and achieve a better score.

4.1 Results

Figure 7 shows how the search proceeded, it shows the population maximum and average plotted against the number of generations.

An analysis of one characteristic and successful member from the run is presented (score of 396.00), it network is graphed in Figure 8.

The graph can be simplified to give:

$$o_{t+1} = \underbrace{81.389 \times i_t}_{\text{Scaled Input}} + \underbrace{0.919 \times o_t}_{\text{Filter}}$$

The same typology of solution was consistently found, it is a simple filter. This cannot be seen to be directly analogous to habituation and sensitisation as I have only used one input.

A more complex filter has not been discovered which suggests a possible weakness in the representation for this problem. E.g. If a filter function

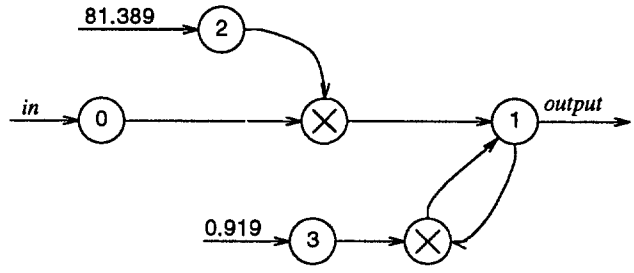


Figure 8: Graphical representation of member 4

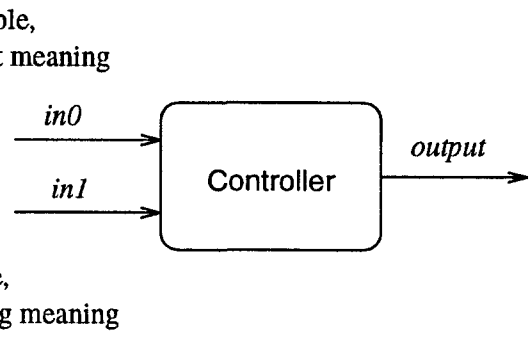


Figure 9: Evolution of associative learning

block was introduced then it would have a greater chance of finding a topology of interconnected filters.

5 Experiment 3: Evolution of associative learning

The third problem to be considered is based on another problem developed by Todd and Miller (Todd & Miller 1991). They created an environment and an evolutionary ANN that supported the emergence of associative learning. The problem is illustrated in Figure 9.

There are two inputs. The first ($in0$) is unreliable, with only a 75% chance of being correct, but its meaning is always the same , a high or a low level had a consistent meaning.

The other input ($in1$) is reliable but its meaning would change between members so that they could not depend on a consistent interpretation, which must be learnt.

Each population member was tested for 1000 trials at three time steps per trial. Its inputs were set for these three time steps before being upda-

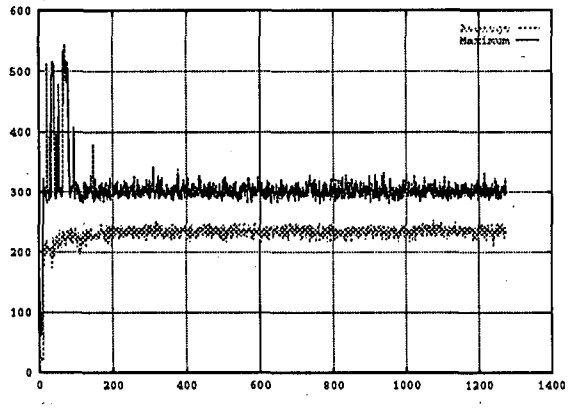


Figure 10: Evolution of associative learning

ted. The inputs were converted to analog values of positive or negative one and clamped to this value. A member had 3 time steps to update its output with the inputs for a trial set.

The output was checked after the inputs had been applied for three time steps. It was converted to a digital value via a threshold function (if it was greater than one then it indicated an output action). The member receives an increase in its score if it acts when the environmental signal is at a high level, and a decrease if it acts at an inappropriate time.

Thus the member had to interpret its two sensors to determine whether to take an action. If it depended only on the unreliable sensor then it would be correct 75% of the time but make mistakes 25% of the time. On the other hand if it relied on the accurate sensor it may not interpret its meaning correctly but has a 50% chance of making a correct interpretation and thus acting with 100% accuracy.

A better solution for a member would be to use the unreliable sensors to learn how to interpretation the accurate sensor, this can be done by correlating the two sensors and filtering.

If a member takes the correct action for the appropriate input state for each of the, on average, 1000/2 rewardable trials then it would obtain an expected score of 500.

5.1 Results

Figure 10 shows how the search proceeded, it shows the population maximum and the average plotted against the number of generations.

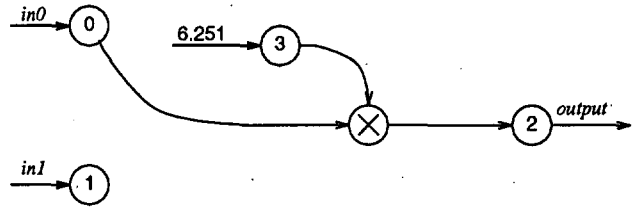


Figure 11: Graphical representation of member 2

After letting this run proceed for 200 generations the simulator was signalled to stop inserting new instruction, and the end state after 1274 generations is shown below.

The code of a typical successful member with a score of 254.00 is presented below, and it is graphed in Figure 11.

The graph can be simplified to:

$$output_{t+1} = 6.251 \times in0_t$$

This experiment failed to find a solution using associative learning. Longer runs of 10000+ generations also failed to find a solution. The solution it found uses the unreliable signal which only has a 75% accuracy, but at least has a consistent meaning.

5.2 Initialised with a solution

In order to verify that a solution does exist and that it is stable a simulation run was performed in which the population was initialised with a known solution. The code used to initialise the population is shown below, and graphed in Figure 12. This member achieved an average score of 500.85.

Figure 13 shows how the search proceeded, it shows the population maximum and average plotted against the number of generations.

After letting this run proceed for 200 generations the simulator was signalled to stop inserting new instruction, and the end state after 808 generations was a slightly different code, a slightly lower average score. This can be accounted for by the random instruction removals and random changes to constants which was still proceeding, which would have created some below average members. The code is essentially the same as the original which at least shows that there was a solution.

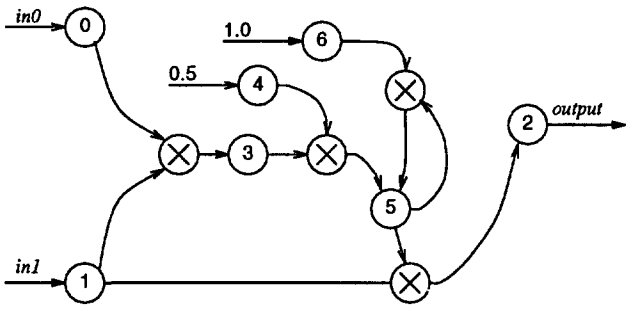


Figure 12: Graphical representation of initial members

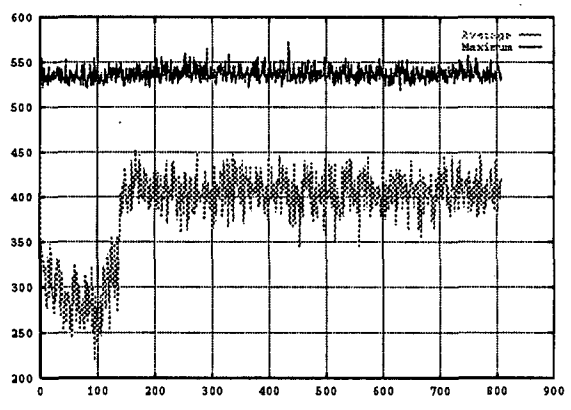


Figure 13: Initialised with a known solution

The code of a typical successful member is presented below, and it is graphed in Figure 14.

6 Discussion and Conclusion

In my investigations of the question of how learning has evolved I aimed to develop a model that allowed a general class of learning algorithms to emerge. I tried to generalise the work of others who had obtained limited success but with very narrow search domains (Miller & Todd 1990, Chalmers 1990). I defined some simple problems and developed an evolutionary algorithm with a search domain of adaptive processes that could result in a useful search while at the same time being relatively general. This was applied with success to the search for a solution to a simple control problem, and to the artificial evolution of habituation and sensitisation, but without

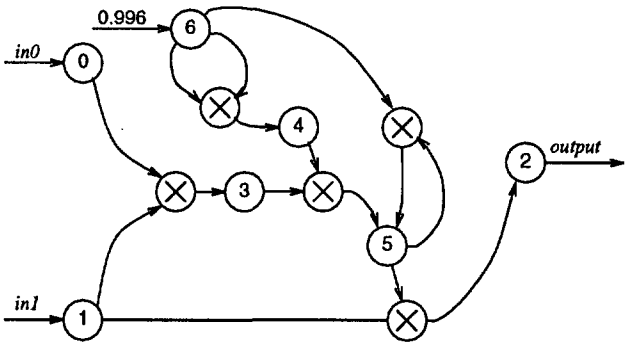


Figure 14: Graphical representation of member 3

success to the problem of the evolution of associative learning. The failure to obtain an associative learning mechanism is explained by the lack of *a priori* knowledge in this experiment compared with that of Todd & Miller.

It is concluded that a learning algorithm will evolve if the designer selects an appropriate *a priori* evolutionary search domain and search technique that is appropriate to the problem which presumably requires learning.

Since solutions to problems requiring learning tend to be rather specific the evolutionary algorithm must be specifically biased in order to find these solutions. In the limit the obvious thing to do is to simply initialise the system with the known answer. In order to solve any specific problem the adaptive process should be as adapted as possible and have as little adaptability as required. Well adapted adaptive processes give the best performance on specific problems.

One way in which *a priori* knowledge can be put into the search algorithm is to initialise the population with an approximate solution. This was shown to offer a trivial solution to the third failed experiment.

In the series of experiments presented it is shown that a trivial solution to the question of how learning mechanisms can evolve, is to start with the solution! If this is considered cheating then you could initialise it to almost the solution so that the solution is easily found. I suggest that in getting a learning mechanism to evolve one must inevitably do just this.

If care is not taken a research program addressing this question can degenerate into examining toy problems with ad hoc solutions.

If you recognise that learning is something specific then the answer to the question of how learning evolves is that it depends on the problem or the selection pressures, and on the state of the evolving system. Thus the question is too general to be of value.

However the question can be sharpened through ammendments and changes to the assumptions, so that answers may make a useful contribution. Some possible approaches are mentioned below.

6.1 Development of adaptive processes to solve problems

If there are constraints on the problem and/or the search technique such that finding a solution becomes difficult, then research could contribute through the development of an adaptive process to solve the problem with the given constraints.

Multiple layers of adaptation would obviously only be used if necessary, so there may be no need for an evolutionary algorithm.

When searching for a solution to a specific problem, if the methodology reduces to iteration (perhaps due to process, controller, or cost function complexity) then an evolutionary algorithm may be a useful heuristic in helping to solve the problem.

Much of the work on the evolution of artificial neural networks (ANN) is justified in similar terms by noting that there are many aspects of an ANN that need tuning and that this can be fruitfully done with genetic algorithms. The designer will be required to define the problem, cost function, search space, the search method, and importantly to speculate that the heuristic will have value.

Unfortunately as the number of ANN parameters to be optimised by the genetic algorithm increases the search space will likely grow to such an extent that the technique becomes practically useless. It was noted by Xin Yao that "Trying to develop a universal representation scheme which can specify any kind of dynamic behaviours of an EANN is clearly impractical, let alone the prohibitive long computation time required to search such a learning rule space." (Yao 1993).

The field of adaptive control has been developing methods for implementing multi-layer adaptive processes to solve particular problems and thus has a similarity to an evolutionary algorithm tuning a learning algorithm. It is interesting to note that it is conventional wisdom in the field of adaptive control (Aström & Wittenmark 1989) that the controller used should be as simple as possible and domain specific information should be used to design specific heuristics to achieve good performance.

6.2 The evolution of learning

If the constraints that have to be placed on the evolutionary search in order for learning to evolve are mappable to some other real problem or to the history of natural evolution then their study may have more then just ad hoc value. These constraints may be something internal to the search such as its current state, or some aspect of the environment which could be seen to lead the search to the solution.

It is argued by Miller and Todd (Todd & Miller 1990) that "learning mechanisms must be understood in terms of their specific adaptive functions", my results support this view.

Many would feel that Humans are proof that complex adaptive processes have emerged through evolution, and an explanation is needed.

Perhaps the human perceptual system has coevolved with its learning system to be able to *understand* learning in ourselves and others and that there may be a reason for the coupling (perhaps a need to be understood both personally and socially).

So perhaps the question needs to be reframed as how and why has the *perception* of a complex capacity to learning evolve? This would loosen the need for a *specific and particular learning problem and solution* which may be just the break needed to develop an explanation? The assumption, in the experiments presented here, that the members of the population are independent would need to be dropped.

There is much literature dealing with these issues of evolution (Monod 1971, Dawkins 1986). The view is that evolution proceeds with constraints imposed on it, but that it also creates and chooses the particular problem(s) which it solves.

If this hypothesis were true then an implementation would only have applicability in so far as the constraints can direct the course of evolution, and by the luck that the problems it chooses to

solve are similar to a real problem. If the path of evolution is determined greatly by chance then re-running it may result in the emergence of a significantly different form of intelligence compared to ours.

Studying such issues may help us understand the environment which shaped the forms of life including its adaptability, how heavily our particular form is dependent on this environment, and thus how general our own perceptual perspective and intelligence really is.

Acknowledgments

This work has benefitted from interactions with Brendan Rogers, Richard Dluzniak, and Alan Bothe who I thank for their encouragement, and support. I also thank the editor Xin Yao and the referees for their comments and suggestions.

The research was partly supported by an APRA-I scholarship.

References

[1] Åström, K. J. & Wittenmark, B. (1989). *Adaptive Control.* Reading, MA: Addison-Wesley.

[2] Chalmers, D. J. (1990). The evolution of learning - An experiment in genetic connectionism. In (Touretzky et. al. 1990).

[3] Dawkins, R. (1986). *The Blind Watchmaker.* Penguin Books.

[4] Fontanari, J. F. & Meir, R. (1991). Evolving a learning algorithm for the binary perceptron. *Network*, 2, p. 353–359.

[5] Forrest, S. & Mitchell, M. (1992). Relative building-block fitness and the building-block hypothesis. In Whitley, D., editor, *Foundations of Genetic Algorithms 2.* San Mateo, CA: Morgan Kaufmann.

[6] Forrest, S. & Mitchell, M. (1993). What makes a problem hard for a genetic algorithm? some anomalous results and their explanation. *Machine Learning.*

[7] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley.

[8] Miller, G. F. & Todd, P. M. (1990). Exploring adaptive agency I - Theory and methods for simulating the evolution of learning. In (Touretzky et. al. 1990), p. 65–80.

[9] Miller, G. F., Todd, P. M., & Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms, Held at the George Mason University, June 4-7, 1989*, p. 379–384. San Mateo, CA: Morgan Kaufmann.

[10] Monod, J. (1971). *Chance and Necessity.* New York: Alfred A. Knopf.

[11] Todd, P. M. & Miller, G. F. (1990). Exploring adaptive agency III - Simulating the evolution of habituation and sensitization. In Schwefel, H. P. & Männer, R., editors, *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, p. 307–313. Berlin: Springer-Verlag.

[12] Todd, P. M. & Miller, G. F. (1991). Exploring adaptive agency II - Simulating the evolution of associative learning. In Meyer, J.-A. & Wilson, S. W., editors, *Proceedings of the First International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, p. 306–315. MIT Press/Bradford Books.

[13] Touretzky, D. S., Elman, J. L., Sejnowski, T. J., & Hinton, G. E., editors (1990). *Proceedings of the 1990 Connectionist Model Summer School.* Sed Mateo, CA: Morgan Kaufmann.

[14] Yao, X. (1993). A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, 8, p. 539–567.

A Preliminary Investigation on Extending Evolutionary Programming to Include Self-Adaptation on Finite State Machines

Lawrence J. Fogel and David B. Fogel
Natural Selection Inc.
1591 Calle De Cinco, La Jolla, CA 92037
AND
Peter J. Angeline
Loral Federal Systems
Rt. 17C, Mail Drop 0210, Owego, NY 13827

Keywords: evolutionary programming, evolutionary computation, self-adaptation

Edited by: Xin Yao
Received: November 16, 1993 **Revised:** June 13, 1994 **Accepted:** June 20, 1994

Evolutionary programming was first offered as an alternative method for generating artificial intelligence. Experiments were offered in which finite state machines were used to predict time series with respect to an arbitrary payoff function. Mutations were imposed on the evolving machines such that each of the possible modes of variation were given equal probability. The current study investigates the use of self-adaptive methods of evolutionary programming on finite state machines. Each machine incorporates a coding for its structure and an additional set of parameters that determine in part how it will distribute new trials. Two methods for accomplishing this self-adaptation are implemented and tested on two simple prediction problems. The results appear to favor the use of such self-adaptive methods.

1 Introduction

Evolutionary computation has a long history (Fogel, 1995, Ch. 3). Some of the first efforts modeled evolution as a genetic process (Fraser, 1957; Bremermann, 1962; Holland, 1975). In these simulations, a population of abstracted chromosomes are modified via operations of crossover, inversion and simple point mutation. An external selection criterion (objective function) is used to determine which chromosomes to maintain as parents for successive generations. These procedures have come to be termed genetic algorithms. Alternatively, Rechenberg (1965) and Schwefel (1965), and also Fogel (1962, 1964), offered methods for simulating evolution as a phenotypic process, that is, a process emphasizing the behavioral link between parents and offspring, rather than their genetic link. These simulations also maintain a population of abstracted organisms (either as individuals or species) but emphasis is placed on the use of mutation operations that generate a continuous range of behavioral diversity yet maintain a strong correlation between the behavior of the parent and its offspring. These methods are known as evolution strategies and evolutionary programming, respectively.

This paper focuses on experiments with evolutionary programming. In particular, self-adaptive parameters that provide information on the generation of offspring are incorporated into evolving solutions and are simultaneously subjected to mutation and selection. Such operations were first offered by researchers in evolution strategies and applied to real-valued function optimization problems, but can be extended to problems in discrete combinatorial optimization. The paper begins with background on evolutionary programming and the use of self-adaptation in evolutionary computation. The results of experiments that compare the efficiency of self-adaptive methods on finite state machines for time series prediction are described. Finally, potential avenues for further investigation are discussed.

2 Background on Evolutionary Programming

Evolutionary programming was originally offered (Fogel, 1962, 1964; Fogel et al., 1966) as a method for generating artificial intelligence (AI). Other attempts to generate AI had been directed toward neural networks (e.g., McCulloch and Pitts, 1943; Rosenblatt, 1958) and heuristic programming (e.g., Simon and Newell, 1958). Each of these avenues focused on modeling or emulating human intelligence, rather than the process that generates intelligent organisms.

Intelligence can be viewed as the ability to adapt behavior to meet goals in a range of environments (Fogel et al., 1966, p. 2; Fogel, 1995). Intelligent behavior requires the composite ability to predict the surrounding environment coupled with a translation of the predictions into a suitable response in light of a given goal. To provide maximum generality, in a series of experiments, Fogel (1964; Fogel et al., 1966) described a simulated environment as a sequence of symbols taken from a finite alphabet. The problem was defined to evolve an algorithm that would operate on the sequence of symbols thus far observed and produce an output symbol that maximizes the benefit to the algorithm in light of the next symbol to appear in the environment and a well-defined payoff function. Finite state machines provided a useful representation for the required behavior.

A finite state machine (Figure 1) is a transducer that can be stimulated by a finite alphabet of input symbols, can respond in a finite alphabet of output symbols, and possesses some finite number of different internal states. The corresponding input-output symbol pairs and next-state transitions for each input symbol, taken over every state, specifies the behavior of any finite state machine, given any starting state. For a review of finite state machines, see Fogel et al. (1966, pp. 149-155).

Evolutionary programming was proposed as operating on finite state machines as follows. A population of "parent" finite state machines is exposed to the environment, that is, the sequence of symbols which have been observed up to the current time. For each parent machine, as each next input symbol is offered to the machine, each output symbol is compared to the next input symbol. The worth of this prediction is then measured with respect to the given payoff function (e.g., all-none, absolute error, squared error, or any other expression of the meaning of the symbols). After the last prediction is made, a function of the payoff for each symbol (e.g., average payoff per symbol) indicates the fitness of the machine.

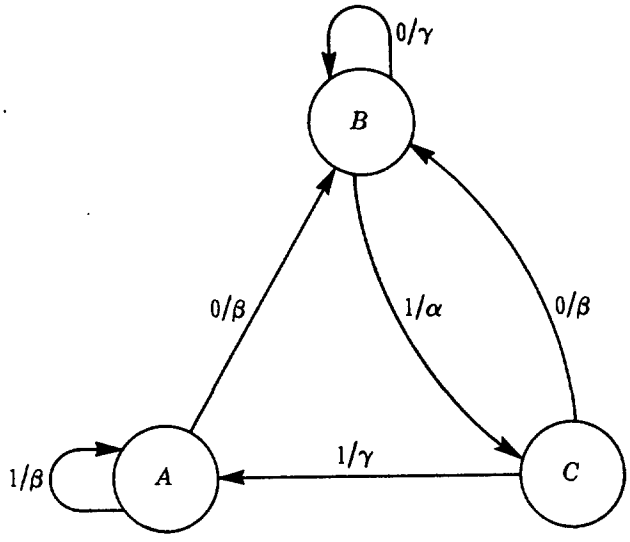


Figure 1: A three-state finite state machine. The input alphabet is {0,1}. The output alphabet is {a, b, g}. Input symbols are shown to the left of the virgule, output symbols to the right.

Offspring machines are created by randomly mutating each parent machine. For convenience, each parent is typically made to produce a single offspring. There are five possible modes of random mutation that naturally result from the description of the machine: change an output symbol, change a state transition, add a state, delete a state, or change the initial state. The deletion of a state and the changing of the initial state are only allowed when the parent machine has more than one state. Mutations are chosen with respect to a probability distribution, which is typically uniform. The number of mutations per offspring is also chosen with respect to a probability distribution (e.g., Poisson) or may be fixed a priori. These offspring are then evaluated over the existing environment in the same manner as their parents. Other mutations, such as majority logic mating, were proposed but results with these mutations were not described in Fogel et al. (1966).

Those machines which provide the greatest

payoff are retained to become parents of the next generation. Typically, half of the total machines are saved so that the parent population remains the same size, but this is not required nor is necessary optimal. This process is iterated until it is required to make an actual prediction of the next symbol (as yet not experienced) in the environment. The "best" machine is chosen to generate this prediction, the new symbol is added to the experienced environment, and the process is repeated. Fogel originally used "nonregressive" evolution. For a machine to be retained it had to score in the top half of the population. Saving lesser adapted machines was discussed as a possibility (Fogel et al., 1966, p. 21) but not incorporated.

There is an inherent versatility in such evolutionary programming. The payoff function can be arbitrarily complex and possess temporal components; there is no requirement for the classical squared error criterion or any other "smooth" function. Further, it is not required that the prediction be made with a one-step look ahead. The prediction of symbols to appear at an arbitrary length of time in the future can be made. Multivariate environments can be handled and the overall environmental process need not be stationary as the simulated evolution will adapt to the changes in the transition statistics.

For example, a nonstationary sequence of symbols was generated by classifying each of the increasing integers as being prime (symbol 1) or nonprime (symbol 0). Thus the environment consisted of the sequence 01101010001... where each symbol depicts the primeness of the positive integers 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,... respectively. The payoff for prediction was all-or-none, that is, one point for each correct prediction, zero points for each error, modified by subtracting 0.01 multiplied by the number of states of the machine. This penalty for complexity was provided to maintain parsimonious machines in light of the limited memory of the available computer (an IBM 7094).

Figure 2 shows the cumulative percentage of correct predictions in the first 200 symbols. After the initial fluctuation (due to the small sample size), the prediction score increased to 78 percent at the 115th symbol and then essentially remained constant until the 200th prediction. At this point, the best machine possessed four states. At the 201st prediction, the best machine possessed three states, and at the 202nd prediction, the best machine possessed only one state with both output symbols being 0. After 719 symbols, the process was halted with the cumulative percentage of correct predictions reaching 81.9 percent. The asymptotic worth of this machine would be 100 percent correct because the prime numbers become increasingly infrequent and the machine continues to predict "nonprime."

The goal was then changed to offer a greater payoff for predicting a rare event. Correctly predicting a prime was worth one plus the number of nonprimes that preceded it. Similarly, correctly predicting a nonprime was worth one plus the number of nonprimes which preceded that nonprime. During the first 150 symbols there were 30 correct predictions of primes, 37 incorrect predictions (false alarms) and five missed primes. From the 151st symbol to the 547th there were 65 correct predictions of primes and 67 false alarms. That is, of the first 35 primes, five were missed; of the next 65 primes, none was missed. Fogel et al. (1966) indicated that the evolutionary algorithm quickly learned to recognize numbers that are divisible by two or three as being nonprime. Some recognition that numbers divisible by five are nonprime was also evidenced. Fogel (1968) later remarked that the evolutionary programming had successively discovered "cyclic properties within the environment... in essence, the equivalent of first recognizing that numbers divisible by two are not prime, etc. In other words, the program was synthesizing a definition of primeness without prior knowledge of the nature of primeness or an ability to divide."

More recently, evolutionary programming has been applied to diverse combinatorial and general function optimization problems. These include the traveling salesman problem (Fogel, 1988, 1993), evolving neural networks (Fogel et al., 1990; McDonnell and Waagen, 1994; Angeline et al., 1994), system identification (Fogel, 1991), automatic control (Sebald and Schlenzig, 1994; Saravanan, 1994), pattern recognition (Bhattacharjya and Roysam, 1994), and others. In many cases, the approaches are very similar to those of evolution strategies (see Davidor et al., 1994) in that the chosen representation follows from the

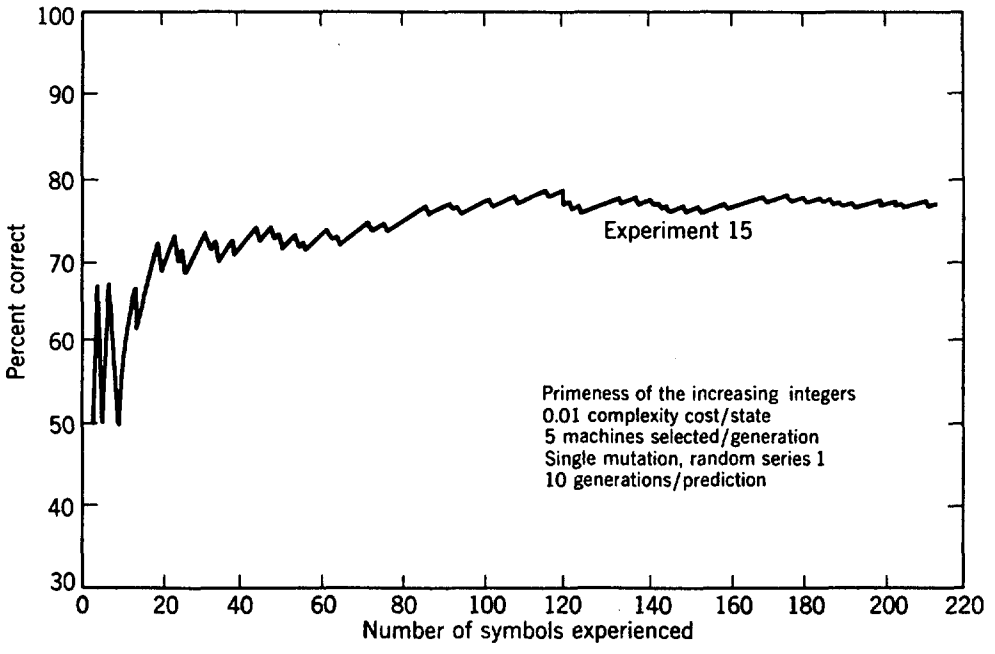


Figure 2: The cumulative percent correct score when using evolutionary programming to design finite state machines to predict the primeness of theincreasing integers (after Fogel et al., 1966).

task at hand and the primary method of searching the space of potential solutions relies on the use of carefully constructed mutation operations that maintain a functional (behavioral) link between parent and offspring. The approaches often differ from those of genetic algorithms (see Davis, 1991) in that no emphasis is placed on the use of crossover or any other operator that would overtly mimic natural genetic mechanisms, and that no effort is made to assess credit to subsections of a solution (cf. schema theory, Holland, 1975). Several direct comparisons of evolutionary programming and evolution strategies to genetic algorithms on benchmark optimization problems have indicated statistically significant evidence favoring the use of the former techniques (Fogel and Atmar, 1990; Bäck and Schwefel, 1993; Bäck, 1994; Fogel, 1994; Fogel and Stayton, 1994; Nettleton and Garigliano, 1994; and others).

3 Incorporating Self-Adaptive Mutation Noise

The ultimate effectiveness of any evolutionary optimization algorithm is determined by the relationship between the shape of the response surface (landscape) being searched and the mutation operations that are used to generate new trial solutions. The rate of optimization may be much greater if the mutative distribution can be tuned to follow grooves and valleys on the surface, rather than simply spray new trials with equal average step sizes in each dimension. The idea for allowing an evolutionary algorithm to self-adapt the manner in which it distributes new trials goes back to Rechenberg in 1967 (Rechenberg, 1994), but was more explicitly detailed in Schwefel (1981).

For example, consider the problem of finding the real-valued n-dimensional vector x that minimizes F(x). Each trial solution is taken to be a pair of vectors (x, σ), where x is the vector of object variables to be assessed by F(x), and σ is a vector of standard deviations (often described as strategy parameters) corresponding to the step sizes of a zero mean multivariate Gaussian random variable. Offspring are created from each parent by the following rules:

$$x_i' = x_i + N(0, \sigma_i)$$

$$\sigma_i' = \sigma_i \cdot \exp(\tau \cdot N(0,1) + \tau' \cdot N_i(0,1))$$

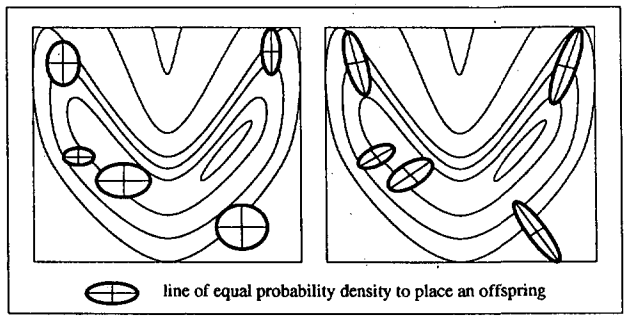


Figure 3: Contour plots of a response surface mapped onto two variable dimensions. Under independent Gaussian perturbations to each component of every parent, new trials are distributed such that the contours of equal probability are aligned with coordinate axes (left picture). This will not be optimal in general because the contours of the response surface are rarely similarly aligned. Schwefel (1981) suggested a mechanism for incorporating self-adaptive covariance terms. Under this procedure, new trial can be distributed in any orientation (right picture). The evolutionary process adapts to the contours of the response surface, distributing trials so as to maximize the probability of discovering improved solutions.

where τ and τ' are operator-set parameters, $N(\mu, \sigma)$ is a normally distributed random variable with mean μ and standard deviation σ, and $N_i(0,1)$ describes a standard Gaussian resampled a new for the ith component of σ. Figure 3 indicates the potential for such a method to distribute trials in relation to the contours of the adaptive landscape. The technique distributes solutions in directions that have provided improved solutions in the past. Schwefel (1981) extended this method to allow for arbitrary correlations between perturbations.

Fogel et al. (1991) independently offered a similar self-adaptive procedure for evolutionary programming in which the standard deviations are altered using a Gaussian random variable. Specifically, the method is:

$$x_i' = x_i + N(0, \sigma_i)$$

$$\sigma_i' = \sigma_i + \alpha \sigma_i N(0,1)$$

where α is a scaling parameter. If any value σ_i' becomes nonpositive, it is reset to a small arbitrary value ϵ. Fogel et al. (1992), at the suggestion of Sebald (1991), incorporated an additional procedure to allow for arbitrary correlations between the strategy parameters. Comparisons by Saravanan and Fogel (1994) indicate that the method of Schwefel (1981) generally outperforms the method of Fogel et al. (1991) when limited to uncorrelated perturbations of the strategy parameters. No comparisons have been made between the methods incorporating complete covariance matrices.

The idea for self-adapting the distribution of new trials also arose independently in genetic algorithms and genetic programming. Schaffer and Morishima (1987) offered a method for self-adapting crossover points. Each binary string encoded not only the n-bit solution vector, but an additional n-bit binary mask that determined the crossover points on the solution vector and was itself subject to mutation. Angeline and Pollack (1992) added mutation operators to a genetic program (Koza, 1992) that protected entire subtrees from both crossover and mutation. Angeline and Pollack (1994) argued that protected subtrees raise the representational level of the primitive language in a task-specific manner.

Recently, Angeline and Pollack (1993) provided a different form of self-adaptation in evolutionary programming as applied to finite state automata in which individual links and output symbols could be randomly "frozen," effectively negating any probability for mutation. The current investigation examines the potential for more gradually affecting the probability of mutating links and output symbols in finite state machines used for time series prediction.

4 Experiments

The base-line method of evolutionary programming investigated was similar to that of Fogel et al. (1966) and refined in Fogel (1991). Each machine in the population was judged in terms of a fitness function which represents the cost and benefit of each possible error or correct prediction. Each machine received a tournament score based on its fitness relative to q other machines selected at random from the population (Fogel, 1991); in each competition, if its fitness was equal to or greater than its opponent, it received a "win."

Parents for the next generation were chosen by ranking the population based on the number of wins (instead of raw fitness) and selecting those individuals scoring in the top half. Each parent created a single offspring in accordance to specific mutation operations.

Five modes of mutation were used to create offspring: add a state, delete a state, change the initial state, change an output symbol, and change a next-state transition. The mutation operation selects a specific mode of mutation for any single manipulation of a machine uniformly across modes. The specific component to modify is chosen in accordance with a uniform distribution from the set of such components in the machine (e.g., if an output symbol is to be changed, each output symbol has an equal chance of being selected). The number of mutations per parent was given by a Poisson random variable with a rate of 3.0. The maximum number of states for any machine was set to 25 and the minimum number of states was set to three. Two self-adaptive methods for evolving finite state machines were examined: selective self-adaptation and multi-mutational self-adaptation.

4.1 Selective Self-Adaptation

In this method of self-adaptation, a mutability parameter was associated with each component of a finite state machine. For each mutation, a component was selected based on its mutability parameters. Specifically, the probability that the ith component was selected was given by:

$$P(i)/\sum P(k)$$

where P_i is the mutability parameter for the ith component, and the summation is taken with the index k running over all components. Separate mutability parameters were maintained for each state (i.e., probability of deleting the state), each output symbol on a transition based on an input symbol, and each next-state transition. For example, if the chosen mutation was to delete a state, the mutability parameters associated with each state of the machine were used to determine the relative probability of deleting each state. Similarly, when the chosen mutation indicated changing an output symbol associated with a transition in the machine, the particular transition was chosen using the mutability parameters associated with the output symbols of the machine's transitions.

All mutability parameters for each machine were initially set to a minimum value of 0.001. Thus each component of any initial machine was equally likely to be selected for mutation at the beginning of any trial. Mutability parameters for components of states subsequently incorporated as a result of an add state mutation were also set to the minimum value. The mutability parameters were themselves mutated in a similar fashion to Fogel et al. (1991), specifically

$$\sigma_i' = \sigma_i + \alpha N(0,1)$$

where σ_i is the parent's mutability parameter for the ith component, σ_i' is the offspring's mutability parameter for the ith component, and $\alpha = 0.01$. Any mutability parameter that fell below the minimum value of 0.001 was reset to the minimum; no upper limit was imposed.

4.2 Multi-mutational Self-Adaptation

In a similar manner as selective self-adaptation, multi-mutational self-adaptation associated a mutability parameter with each component of each machine. But in contrast, each mutability parameter designated the absolute probability of modification for that particular component. Thus the probability for each component to be mutated was independent of the probabilities of other components to be mutated, this offering greater diversity in the types of offspring machines that could be generated from a parent. For each offspring, each mutability parameter was compared to the outcome of a uniform random variable on (0,1) (denoted U(0,1)). If the random number was lower than the mutability parameter, the appropriate mutation was executed. For example, mutation would delete each state for which the outcome of the U(0,1) fell below that state's mutability parameter. Similarly, each output symbol and next-state transition were mutated when the resampled U(0,1) fell below the associated mutability parameter. The creation of an offspring was completed after each component of the parent had been tested.

Multi-mutational self-adaptation also modified the mutability parameters using the same technique and standard deviation as with selective

self-adaptation. Unlike selective self-adaptation in which the chosen standard deviation is of little importance because the probabilities of specific mutations are all relative to other mutations, the standard deviation of the Gaussian noise is extremely important under the multi-mutational approach. Given too large a variance, the mutability parameters can decrease the stability and potential evolvability of the resulting offspring. For the current study, the minimum value for a mutability parameter in multi-mutational self-adaptation was set to 0.005. Thus no component had less than a 1 in 200 chance of being modified at any time. If adding Gaussian noise to the parameter resulted in a value less than this minimum it was reset to 0.005. The maximum value for the parameter was set to 0.999. Initial values for the parameters of machines in the initial population and for components added to machines by an add state mutation were set to the minimum value. This ensured that newly added components were initially stable and had to evolve increased mutability.

To offset the potential increase in the deletion rate of states in evolving machines, the probability of adding a state to an offspring was increased to 0.3. The chance of mutating the initial state of a machine in multi-mutational self-adaptation remained at 0.2.

4.3 Design

The above methods were tested on two simple prediction tasks. The first was offered in Fogel et al. (1966). A base string of symbols served as an initial observation from an environment. The environment was taken to be the string (101110011101)*. Fitness was assessed as the fraction of correct predictions made over all observed symbols. A new symbol was introduced into the environment every five generations (i.e., a complete iteration of mutation, competition, and selection). Ten symbols were provided as the initial set of observations. The second environment was taken to be the string (101100111000110010)*. For each environment, the population size was 100 machines and trials were executed over 750 generations. Each experiment consisted of 50 trials with the basic evolutionary program (i.e., all modes of mutation having equal probability, all specific components having equal probability), the selective self-adaptation method and the multi-mutational self-adaptation method.

5 Results

Figure 4 indicates the score of the best machine in the population at each generation averaged over all 50 trials with each of the three methods on the environment (101110011101)*. The curves demonstrate an asymptotic rise toward 100 percent correct, as the cyclic pattern in the environment is mapped by successively better finite state machines. But the rate of improvement across the three methods appears to favor the self-adaptive methods, and more clearly the multi-mutational self-adaptation. Figure 5 shows the t-test score comparing both self-adaptive methods to the basic evolutionary programming. Although there appears to be significant evidence of an improvement with the self-adaptive methods, caution must be used when interpreting these data because they represent a sequence of dependent trials. Figure 6 indicates the score of the best machine in the population at each generation averaged over all trials with each method on the environment (101100111000110010)*. The results are similar to those depicted in Figure 4. Figure 7 indicates the relevant t-scores comparing the self-adaptive methods with the base-line method for this more complex environment.

6 Discussion

The practicality of evolutionary optimization algorithms can be significantly increased through the incorporation of self-adaptive parameters that determine how each parent will distribute future trials (Bäck and Schwefel, 1993). Including such parameters frees the human operator from having to select mutation distributions (or genetic operators in genetic algorithms) ad hoc. The majority of efforts in self-adaptation have pertained to real-valued continuous function optimization problems (Schwefel, 1981; Bäck and Schwefel, 1981; Fogel et al., 1991; Saravanan and Fogel, 1994), but they can be extended to discrete combinatorial optimization problems such as the evolution of finite state machines for time series prediction.

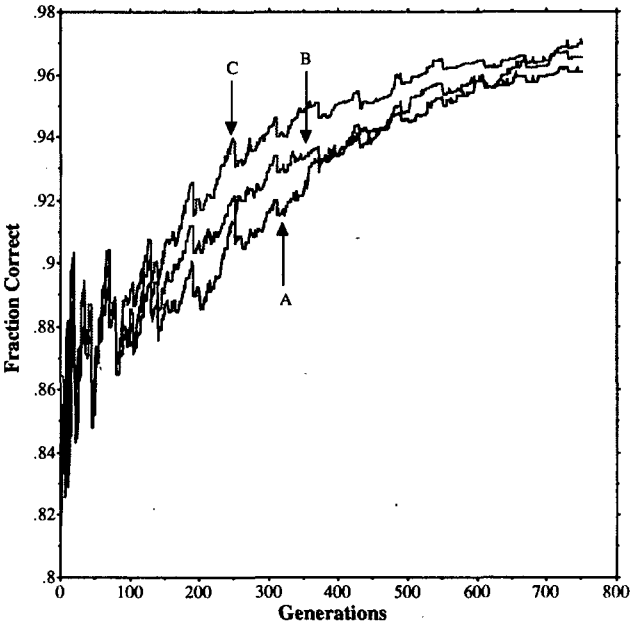


Figure 4: The fraction correct of the best machine in the population at each generation averaged over all 50 trials with each method applied to the environment (101110011101)*. (A) No self-adaptation. (B) Selective self-adaptation. (C) Multi-mutational self-adaptation. Both self-adaptive methods appear to be at least as efficient as (B) or more efficient than (C) the evolutionary program without self-adaptation on the chosen environment.

Self-adaptation on continuous representations allows for parents to continue to generate offspring in directions on the adaptive landscape (error surface) that have proved useful in the past. It essentially serves as a memory of previous trajectories; those that have worked well recently are reinforced while those that have not generated useful trial solutions are purged from the population. But "direction" is difficult to apply to discrete representations. Although it might be useful in some particular real-valued continuous optimization problem to iteratively increase the value of a certain parameter (e.g., move toward increasingly greater values of x while searching for the minimum of f(x)), it is not, by analogy, useful to continue changing an output symbol or next-state transition if such changes have been of value in the past (cf. Lenat, 1983).

Self-adaptation has proved useful on discrete structures (e.g., finite state machines) when a possibility for freezing parameters has been included (Angeline and Pollack, 1993), prohibiting

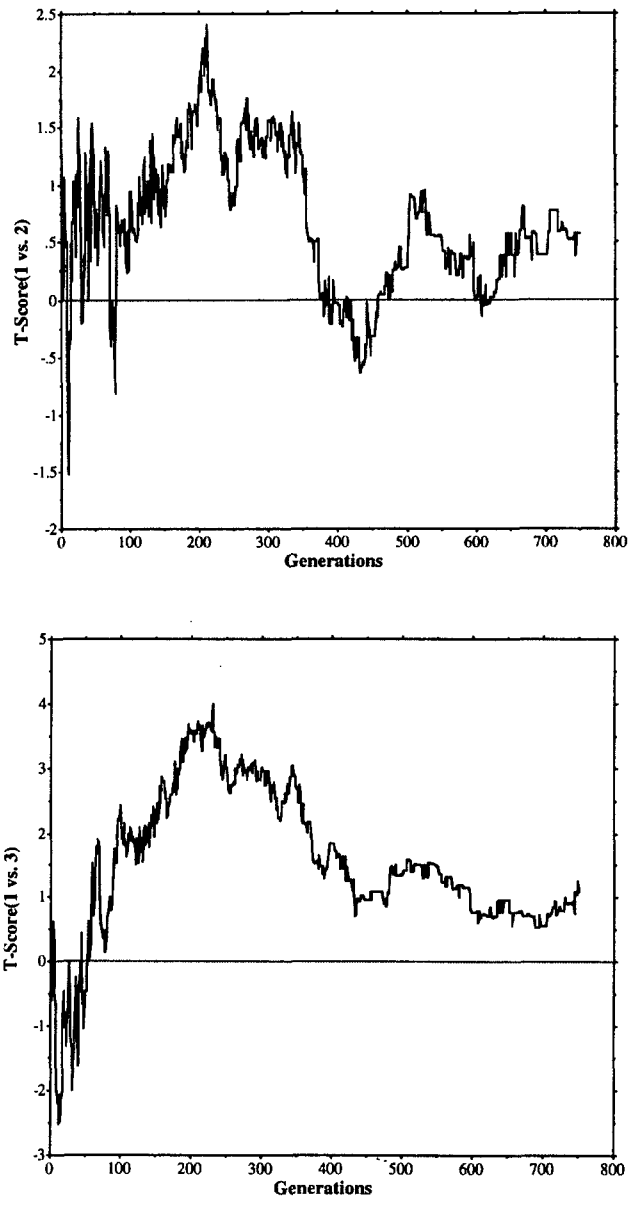


Figure 5: Consecutive t-test scores comparing the results of each self-adaptive method to the baseline results without any self-adaptation on the environment (101110011101)*. (a) Selective self-adaptation vs. no self-adaptation. (b) Multi-mutational self-adaptation vs. no self-adaptation. Positive scores favor the self-adaptive methods while negative scores favor the method without self-adaptation. A typical critical score would be approximately 1.96 under a level of significance of a = 0.05, but the t-scores across generations are correlated and thus no definitive statistical conclusion can be firmly stated. The results do justify an expectation that further analysis will indicate statistically significant differences in favor of the self-adaptive methods.

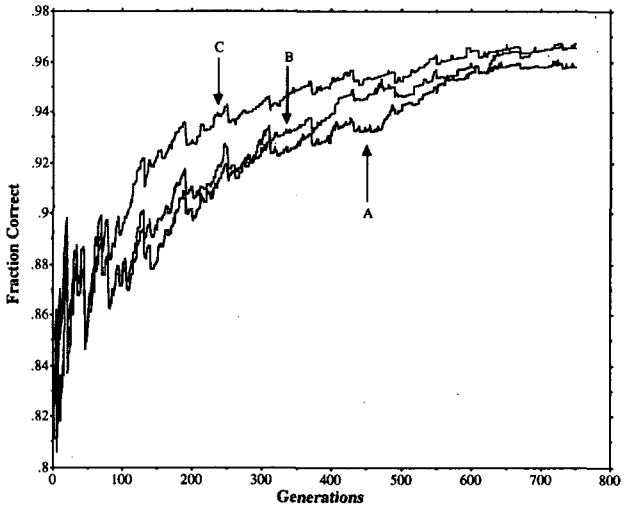


Figure 6: The fraction correct of the best machine in the population at each generation averaged over all 50 trials with each method applied to the environment (101110011101)*. (A) No self-adaptation. (B) Selective self-adaptation. (C) Multi-mutational self-adaptation. Both self-adaptive methods appear to be at least as efficient as (B) or more efficient than (C) the evolutionary program without self-adaptation on the chosen environment.

mutation to certain components and thereby maintaining informational gains held within the coding structure. Rather than mandating either the extreme of completely freezing parameters or the extreme of mutating all parameters with equal probability, the self-adaptive methods examined in the current investigation can transition between these extremes. In essence, the methods allow for a gradual freezing of useful input-output and next-state transitions.

The efficiency of any evolutionary optimization algorithm is directly dependent on the shape of the adaptive landscape being searched and the mutation operations that are used to search the state space. It is crucial that there be a strong functional relationship between each parent and its offspring, while simultaneously offering the potential for nearly continuous functional diversity (Fogel, 1988; and others). This can often be accomplished by the use of zero mean multivariate Gaussian mutations on real-valued function optimization problems (Fogel and Atmar,1990; Bäck

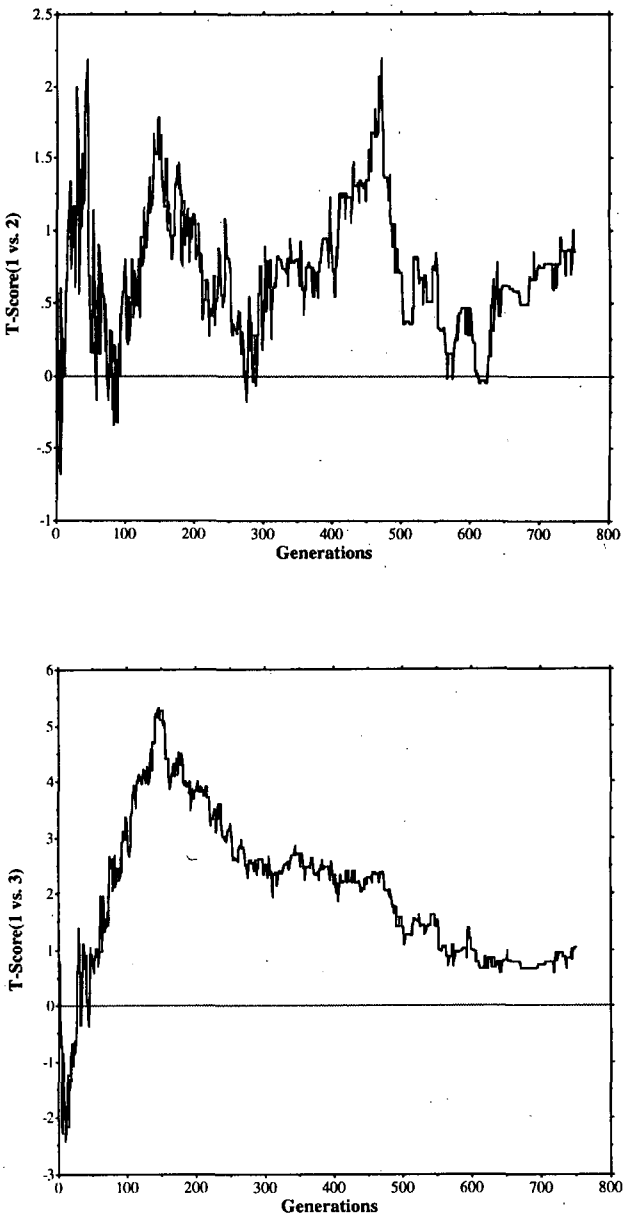


Figure 7: Consecutive t-test scores comparing the results of each self-adaptive method to the baseline results without any self-adaptation on the environment (101100111000110010)*. (a) Selective self-adaptation vs. no self-adaptation. (b) Multi-mutational self-adaptation vs. no self-adaptation. Positive scores favor the self-adaptive methods while negative scores favor the method without self-adaptation. See Figure 5 for a discussion of the interpretation of these data.

and Schwefel, 1993; Fogel and Stayton, 1994; and others). Maintaining functional links between parents and offspring when using discrete representations is more difficult. The proposed self-adaptive methods may provide a general mechanism for achieving this end. The preliminary results appear to indicate improved convergence rates when using either self-adaptive method as compared to failing to use any such method. Moreover, the multi-mutation mechanism in which each parameter is given its own probability of mutation yielded the best results. More careful assessment of the statistical significance of these results, extensions to more complex environments and comparisons between the realized mutation variance (i.e., the mean number of imposed mutations per machine) remain for further investigation.

7 Acknowledgments

The authors would like to thank Dr. X. Yao for both his patience and encouragement to participate in this special issue.

References

[1] P.J. Angeline and J.B. Pollack (1992) "The Evolutionary Induction of Subroutines," Proceedings of the 14th Annual Conference of the Cognitive Science Society, Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, pp. 236-241.

[2] P.J. Angeline and J. Pollack (1993) "Evolutionary Module Acquisition," Proceedings of the Second Annual Conference on Evolutionary Programming, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 154-163.

[3] P.J. Angeline and J.B. Pollack (1994) "Coevolving High-Level Representations," Artificial Life III, C.G. Langton (ed.), Addison-Wesley, Reading, MA, pp. 55-71.

[4] P.J. Angeline, G.M. Saunders and J.B. Pollack (1994) "An Evolutionary Algorithm That Constructs Recurrent Neural Networks," IEEE Trans. Neural Networks, Vol. 5:1, pp. 54-65.

[5] T. Bäck (1994) Evolutioary Algorithms in Theory and Practice, IOP Press, Philadelphia, PA, in press.

[6] T. Bäck and H. -P. Schwefel (1993) "An Overview of Evolutionary Algorithms for Parameter Optimization," Evolutionary Computation, Vol. 1:1, pp. 1-24.

[7] A.K. Bhattacharjya and B. Roysam (1994) "Joint Solution of Low-, Intermediate-, and High-Level Vision Tasks by Evolutionary Optimization: Application to Computer Vision at Low SNR," IEEE Trans. Neural Networks, Vol. 5:1, pp. 83-95.

[8] H.J. Bremermann (1962) "Optimization through Evolution and Recombination," Self-Organizing Systems, M.C. Yovits, G.T. Jacobi, and G.D. Goldstine (eds.), Spartan Books, Washington D.C., pp. 93-106.

[9] Y. Davidor, H.-P. Schwefel and R. Männer (eds.) (1994) Parallel Problem Solving from Nature 3, Springer-Verlag, Berlin.

[10] L. Davis (ed.) (1991) Handbook of Genetic Algorithms, Van Nostrand Reinhold, NY.

[11] D.B. Fogel (1988) "An Evolutionary Approach to the Traveling Salesman Problem," Biological Cybernetics, Vol. 60:2, pp. 139-144.

[12] D.B. Fogel (1991) System Identification through Simulated Evolution: A Machine Learning Approach to Modeling, Ginn Press, Needham, MA.

[13] D.B. Fogel (1993) "Applying Evolutionary Programming to Selected Traveling Salesman Problems," Cybernetics and Systems, Vol. 24, pp. 27-36

[14] D.B. Fogel (1994) "Asymptotic Convergence Properties of Genetic Algorithms and Evolutionary Programming: Analysis and Experiments," Cybernetics and Systems, Vol. 25:3, pp. 389-407.

[15] D.B. Fogel (1995) Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, IEEE Press, Piscataway, NJ, in press.

[16] D.B. Fogel and J.W. Atmar (1990) "Comparing Genetic Operators with Gaussian Mutations in Simulated Evolutionary Processes Using Linear Systems," Biological Cybernetics, Vol. 63, pp. 111-114.

[17] D.B. Fogel, L.J. Fogel and J.W. Atmar (1991) "Meta-Evolutionary Programming," Proc. of the 25th Asilomar Conference on Signals, Systems and Computers, R.R. Chen (ed.), Maple Press, San Jose, CA, pp. 540-545.

[18] D.B. Fogel, L.J. Fogel, W. Atmar and G.B. Fogel (1992) "Hierarchic Methods of Evolutionary Programming," Proceedings of the First Annual Conference on Evolutionary Programming, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 175-182.

[19] D.B. Fogel, L.J. Fogel and V.W. Porto (1990) "Evolving Neural Networks," Biological Cybernetics, Vol. 63, pp. 487-493.

[20] D.B. Fogel and L.C. Stayton (1994) "On the Effectiveness of Crossover in Simulated Evolutionary Optimization," BioSystems, Vol. 32:3, pp. 171-182.

[21] L.J. Fogel (1962) "Autonomous Automata," Industrial Research, Vol. 4:2, pp. 14-19.

[22] L.J. Fogel (1964) "On the Organization of Intellect," Doctoral Dissertation, UCLA.

[23] L.J. Fogel (1968) "Extending Communication and Control through Simulated Evolution," Bioengineering An Engineering View, Proc. Symp. Engineering Significance of the Biological Sciences, G. Bugliarello (ed.), San Francisco Press, San Francisco, CA, pp. 286-304.

[24] L.J. Fogel, A.J. Owens and M.J. Walsh (1966) Artificial Intelligence through Simulated Evolution, John Wiley, NY.

[25] A.S. Fraser (1957) "Simulation of Genetic Systems by Automatic Digital Computers. I. Introduction," Australian Journal of Biological Sciences, Vol. 10, pp. 484-491.

[26] J.H. Holland (1975) Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI.

[27] J.R. Koza (1992) Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA.

[28] D.B. Lenat (1983) "The Role of Heuristics in Learning by Discovery: Three Case Studies," Machine Learning, R.S. Michalski, J.G. Carbonell, T.M. Mitchell (eds.), Tioga Publishing, Palo Alto, CA, pp. 243-306.

[29] W.S. McCulloch and W. Pitts (1943) "A Logical Calculus of the Ideas Immanent in Nervous Activity," Bulletin of Mathematics and Biophysics, Vol. 5, pp. 115-123.

[30] J.R. McDonnell and D. Waagen (1994) "Evolving Recurrent Perceptrons for Time-Series Modeling," IEEE Trans. Neural Networks, Vol. 5:1, pp. 24-38.

[31] D.J. Nettleton and R. Garigliano (1994) "Evolutionary Algorithms and a Fractal Inverse Problem," BioSystems, in press.

[32] I. Rechenberg (1965) "Cybernetic Solution Path of an Experimental Problem," Royal Aircraft Establishment, Library Translation 1122, Farnborough, Hants, U.K.

[33] I. Rechenberg (1994) personal communication, Technical University of Berlin, Germany.

[34] F. Rosenblatt (1958) "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," Psychological Review, Vol. 65, p. 386.

[35] N. Saravanan (1994) "Reinforcement Learning Using Evolutionary Programming," Proceedings of the Third Annual Conference on Evolutionary Programming, A.V. Sebald and L.J. Fogel (eds.), World Scientific Publishing, River Edge, NJ, in press.

[36] N. Saravanan and D.B. Fogel (1994) "Learning Strategy Parameters in Evolutionary Programming: An Empirical Study," Proceedings of the Third Annual Conference on

Evolutionary Programming, A.V. Sebald and L.J. Fogel (eds.), World Scientific Publishing, River Edge, NJ, pp. 175-184.

[37] J.D. Schaffer and A. Morishima (1987) "An Adaptive Crossover Distribution Mechanism for Genetic Algorithms," Proc. of the Second International Conference on Genetic Algorithms, J.J. Grefenstette (ed.), Lawrence Erlbaum, Hillsdale, NJ, pp. 36-40.

[38] H.-P. Schwefel (1965) "Kybernetische Evolution als Strategie der Experimentellen Forschungin der Strömungstechnik," Diploma Thesis, Technical University of Berlin.

[39] H.-P. Schwefel (1981) Numerical Optimization of Computer Models, John Wiley, Chichester, U.K.

[40] A.V. Sebald (1991) personal communication, UCSD.

[41] A.V. Sebald and J. Schlenzig (1994) "Minimax Design of Neural Net Controllers for Highly Uncertain Plants," IEEE Trans. Neural Networks, Vol. 5:1, pp. 73-82.

[42] H.A. Simon and A. Newell (1958) "Heuristic Problem Solving: The Next Advance in Operations Research," Operations Research, Vol. 6, p. 6.

Analysis and Comparisons of Genetic Algorithm, Simulated Annealing, TABU Search, and Evolutionary Combination Algorithm

Takashi Kido, Kazuo Takagi and Masakazu Nakanishi
Faculty of Science and Technology, Keio University
3-14-1, Hiyoshi, Kohoku-ku, Yokohama, 223, Japan
kido@math.keio.ac.jp, takagi@math.keio.ac.jp, czl@math.keio.ac.jp

Keywords: genetic algorithms, hybrid search

Edited by: Xin Yao
Received: November 16, 1993 **Revised:** August 20, 1994 **Accepted:** August 27, 1994

This paper describes a hybrid search scheme for genetic algorithms (GAs). Since GA's weakness in local search is well-known, many GA applications combine genetic algorithms with a local search scheme. We have discovered that solution quality and stability can be improved further by using multiple local search strategies with GAs. In particular, we have combined GAs with two major local search mechanisms: TABU search and simulated annealing (SA). We have tested our approach (GA+SA+TABU) using a 100-city traveling salesman problem (TSP). The results indicate that solution quality and stability are superior than those of the GA, SA, or TABU alone.

1 Introduction

GAs are well known for their weakness in local search. To overcome this, many real applications of GAs use domain-specific local search mechanisms [8]. This paper examines hybridization of GAs and local search mechanisms, and proposes the combination of multiple local search algorithms and GAs as a general hybrid GA architecture. In particular, we use TABU search and Simulated Annealing (SA) as the local search mechanisms in this paper. As a benchmark, we use a typical combinatorial problem — a 100-city TSP.

While several algorithms for solving combinatorial optimization problems exist, only a handful of studies have been made on general hybrid methods. Malek proposed a hybrid technique for improving solution quality by mixing two or more algorithms and obtained improved results for TSPs [4]. Malek's idea was to execute each low level algorithm for some specified time, and have the results evaluated by a high level algorithm which then restarts the low level routines in more promising areas in the solution space. In their work, the high level algorithm selects best local solutions from the local search, and uses the results as starting points for the next iteration. However, because Malek's method uses only the best solution from the previous search, it does not maintain global sampling and is vulnerable to local minima.

A different algorithm, genetic annealing, proposed by Yao [18] uses GAs as the high level algorithm and SA as the low level one. The results generated by SA are subject to crossover and mutation.

We have extended Malek's model to maintain diversity by using a GA as the high level algorithm. We also introduce multiple local search mechanisms so that sampling point diversity can be maintained even with the different convergence characteristics of each local search scheme.

First, we describe GAs and local search. We point out a weakness of GA and give an explanation of the simulated annealing (SA) and tabu search (TABU) algorithms as local search algorithms. Second, we propose the GA+SA+TABU method which combines a GA with SA and TABU.

Finally, we examine the convergence characteristics of the GA, SA, TABU, GA+SA, GA+TABU, and GA+SA+TABU. Results indicate that solution quality and stability with this

approach are superior than those of the GA, SA, TABU, GA+SA, or GA+TABU.

2 An outline of the Traveling Salesman Problem

The traveling salesman problem (TSP) is a typical NP-complete problem. It is easy to describe, but difficult to solve. A salesman, starting from his home city, is to visit each city once and only once in a given list and then return home. The problem is to find the tour that minimizes the total distance. Mathematically, given a sequence of cities $c_1, c_2, \cdots, c_n$ and intercity distances $d(c_i, c_j)$, the permutation π must be found that minimizes the sum of distances.

$$\sum_{i=1}^{n-1} d(C_\pi(i), C_\pi(i+1)) + d(C_\pi(n), C_\pi(1))$$

There are many practical applications of the TSP in the real world. Thus finding an effective method to solve the TSP is very important. In this paper, we focus on the case of symmetric TSPs, where the distances satisfy the condition $d(c_i, c_j) = d(c_j, c_i)$ for $1 \leq i, j \leq n$.

The TSP has been approached by many researchers. Lawler et al. have summarized the experimental and theoretical issues around TSPs [1]. There are exact, heuristic, probabilistic methods to solve TSPs. Exact methods include cutting planes, branch and bound, and dynamic programming. However, because the TSP is NP-complete, exact methods are only able to solve small problems without specialized problem reduction. Heuristic and probabilistic methods are able to solve large problems. Some examples of these methods include 2-opt, markov chains, TABU Search, neural networks, simulated annealing, and genetic algorithms.

A 532-city problem was solved to optimality by Padberg and Rinaldi using combination of problem reduction, cutting planes, and branch and bound [12]. Johnson provided optimal solutions for several selected problems from the literature using an iterated Lin-Kernighan algorithm [13]. Malek reported that TABU search and a version of simulated annealing exhibited similar performance [4]. Among the most promising GA results are those of Muhlenbein where the solution length of 27702 for the 532-city problem is close to the known optimal length of 27686 [16].

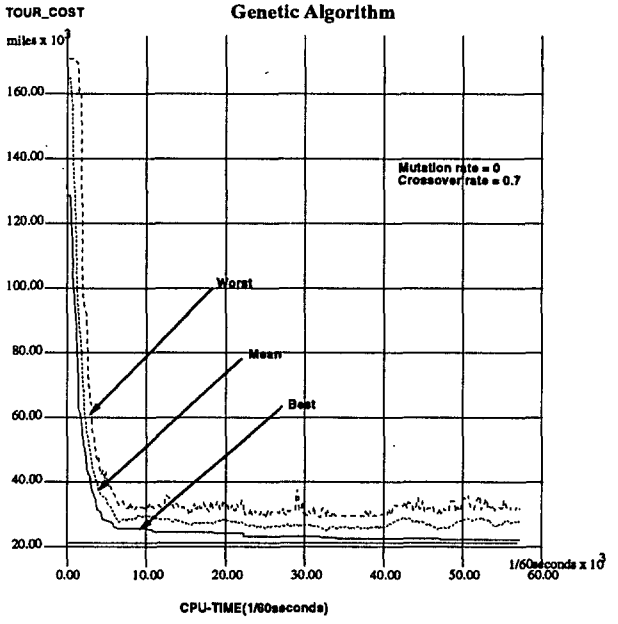


Figure 1: Tour length for 100-city TSP by GA

3 Genetic Algorithms and Local Search

3.1 Genetic Algorithms

Although GAs exhibit very fast convergence to an approximate solution in a search space, a genetic algorithm itself does not include a local search mechanism. When a population reaches a state where it is dominated by the best chromosome, finding better genetic solutions requires mutations. This would result in a very inefficient search.

This is clearly seen in Figure 1, which shows the tour length at each generation.

In this paper, we use the TSP to compare the effectiveness of each approach. We chose the TSP because it has been extensively investigated by various researchers in the GA community (such as [2]), and optimization research groups. We employ path representation encoding. The fitness of

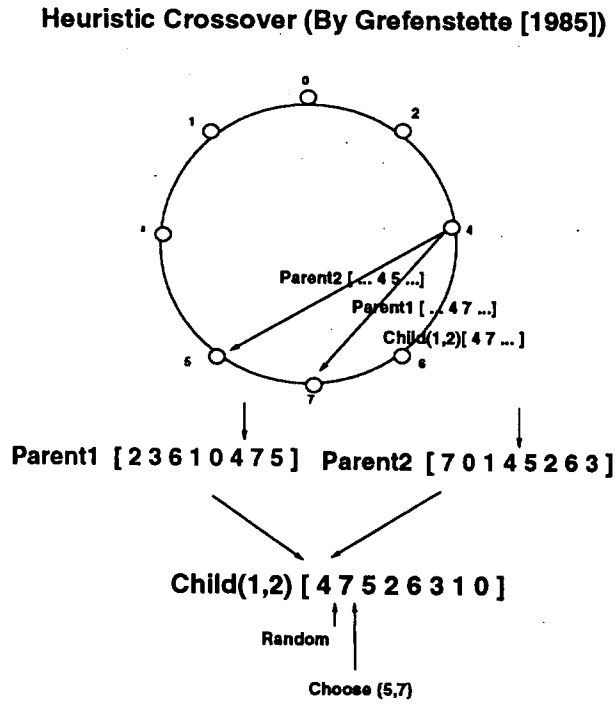


Figure 2: Heuristic greedy crossover

a chromosome is defined as:

$$Fitness = \frac{1}{Tourlength}$$

The reproduction strategy uses a proportional reproduction scheme. In addition, we adopt elitist reproduction which always chooses the best chromosome and copied it, without crossover or mutation, to the next generation. We use the greedy crossover presented by [11] (Figure 2). This operator constructs offspring from two parent tours as follows: Pick a random city as the starting point for the child's tour. Compare the two edges leaving the starting city in the parent's tours, and choose the shorter edge. Continue to extend the partial tour by choosing the shorter of the two edges in the parent's tours which extend the tour. If the shorter parental edge introduces a cycle into the partial tour, then extend the tour by a random edge. Continue until a complete tour is generated. We used a crossover probability of 70.0%, and mutation probability of 0%. We have raised mutation rate up to 5%, but no significant change was observed.

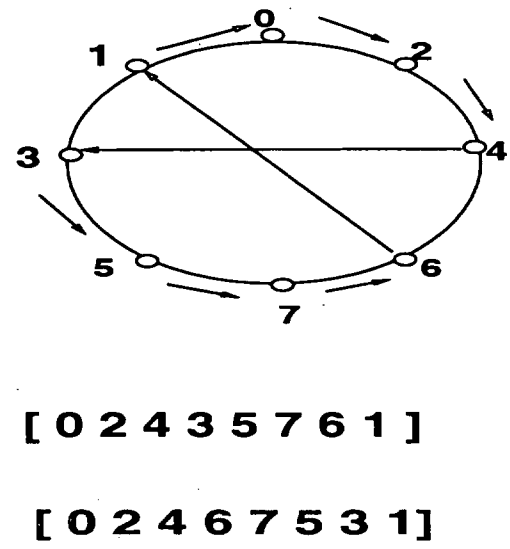


Figure 3: 2-opt move

3.2 Local Search Techniques

3.2.1 Simulated Annealing (SA)

SA is an efficient stochastic search inspired by the physical annealing analogy [3]. To avoid being trapped in local minima, SA moves probablistically and allows uphill movement with probability $exp(-\delta C/T)$, where δC is the uphill cost and T (temperature)is the control parameter.

Annealing performance varies greatly depending upon the changing mechanism used [17]. We use the same changing mechanism and implementation as Malek. The main body of the algorithm consists of two loops, with one nested within the other. The inner loop runs until a quasi-equilibrium is reached. In this loop, possible moves are generated using 2-opt exchange (Figure 3) and the accept decision is made using a function call. Basically, a 2-opt move (swap) exchanges two non-adjacent edges. In Figure 3, we delete the edges between nodes 1 and 6 and between 4 and 3. We replace them with edges between nodes 1 and 3 and between 4 and 6. In path representation encoding, the 2-opt swap can then be performed simply by reversing the order of all the cities in the tour from node 6 to node 3.

If the move is accepted, it is applied to the current tour to generate the next state. Equilibrium is reached when large swings in energy (tour length) no longer occur.

The outer loop checks if the stopping condition

has been met. Each time the inner loop is completed, temperature T is updated and the stopping criterion is checked.

The accept function

```
if(dc < 0) return(true)
elseif(exp(-dc/T) <= random(0, 1))
  return(true)
else return(false)
```

A random number is used to test whether the move is accepted. In our implementation of the simulated annealing algorithm, we choose the stopping criteria to be a temperature such that the probability of accepting an uphill move is very close to zero. After a fixed number of iterations we assume equilibrium is reached. Finally, to update temperature following the equilibrium, we simply multiply the current temperature by a constant α. These parameters allow the algorithm to be tuned for the TSP.

```
WHILE(stopping criterion not met)
  WHILE(equilibrium not reached)
    Generate-next-move()
    IF(Accept(Temperature, change-in-cost))
      Update-tour()
  ENDWHILE
  Calculate-new-temperature()
ENDWHILE
```

To implement the simulated annealing algorithm, as described earlier, the stopping and equilibrium criteria, and the update temperature rule must be specified. The stopping criterion chosen for the algorithm is for the temperature to reach a specified value. This stopping temperature is chosen such that the probability of accepting an uphill move is very close to 0. We make the typical assumption that the equilibrium is reached after a fixed number of iterations. The update rule is

$$NewTemperature = \alpha * Temperature$$

where α is a constant less than one. The consequence of choosing simple constants as parameters is some increase in computation time. It forces the choice of α and the number of iterations to be tuned for critical temperature regions. Such regions require a slow annealing rates. It is possible, however, at high and low temperatures to anneal at faster rates.

As inputs, our SA algorithm implemented has the initial temperature, the number of iterations to simulate equilibrium, and α. These parameters allow the algorithm to be tuned for any TSPs.

3.2.2 TABU Search

Tabu search is another optimization technique for solving permutation problems [14] [15]. In this technique, we start with an arbitrary permutation and make a succession of moves to make this permutation optimal (or as close to the optimum as possible). In determining the shortest tour for a given set of cities, the tabu search procedure starts with a randomly generated tour and makes a succession of 2-opt exchanges that reduce the cost. At each step, all possible 2-opt moves are examined and the one which gives the best improvement in tour cost is chosen. To prevent the process from being trapped at a local optimum, this algorithm allows moves that increase the tour cost (uphill moves). It is more than likely that the moves succeeding an uphill move will turn back to the local optimum. To avoid cycling, the procedure maintains a history of recent moves and classifies such moves as tabu. This enables the search process to escape local optima and explore new areas of the solution state space.

Creating a tabu classification for the moves means identifying swap attributes which could indicate one of the following:

the cities involved in the swap, or

the positions they occupy before/after the swap, or

the direction in which the cities move in the swap.

The tour of all cities is represented in a one-dimensional array format, with the array index denoting the position of the city in the tour. If the city moves from a lower to a higher index during a swap, then it is said to move right. Conversely, if it moves from a higher index to a lower one, then it is said to move left.

We also need to identify the tabu classifications based on the attributes so that we can specify a set of moves as tabu. These attributes are discussed in detail later. Figure 4 shows the tabu search strategy superimposed on the hill climbing heuristic.

The algorithm examines all the swaps of the current tour and keeps track of the best-swap-

value. However, those that are classified as tabu are rejected if they do not satisfy the aspiration criteria. In other words, we restrict the set of available swaps. The tabu status of the move is overridden and the move is accepted if the swap-value satisfies the aspiration level. The best-swap among all the available swaps for the current tour is obtained at the exit of the inner loop. In the hill climbing method, the best-swap-value is usually negative indicating a reduction in the current tour length. When it becomes positive, the process has reached its termination condition.

In tabu search, the best swap is executed regardless of the sign of the best-swap-value. The best swap from the inner loop is accepted even if it results in a higher tour length. This helps the process climb out of local optima. The outer loop keeps track of the best tour and its length. The tabu list is also updated by including the current move made. The stopping criteria is usually a fixed number of iterations or a fixed computation time specified in the input.

The following are examples of the move attributes and the tabu restrictions based on these attributes. In our implementation, we select only one tabu condition for a specific tabu search process.

Tabu Conditions

1. Vector(I<J<POSITION(I)<POSITION(J))
   this vector is maintained to prevent any future swaps from resulting in a tour with cities I and J occupying POSITION(I) **and** position(J) respectively.

2. Vector(I, J, POSITION(I), POSITION(J))
   the same vector to prevent a swap resulting in city I occupying POSITION(I) **or** city j occupying POSITION(J).

3. Vector(I, POSITION(I))
   to prevent city I from returning to POSITION(I).

4. CITY I
   to prevent city I from moving LEFT of current position.

5. City I
   to prevent city I from moving in any direction.

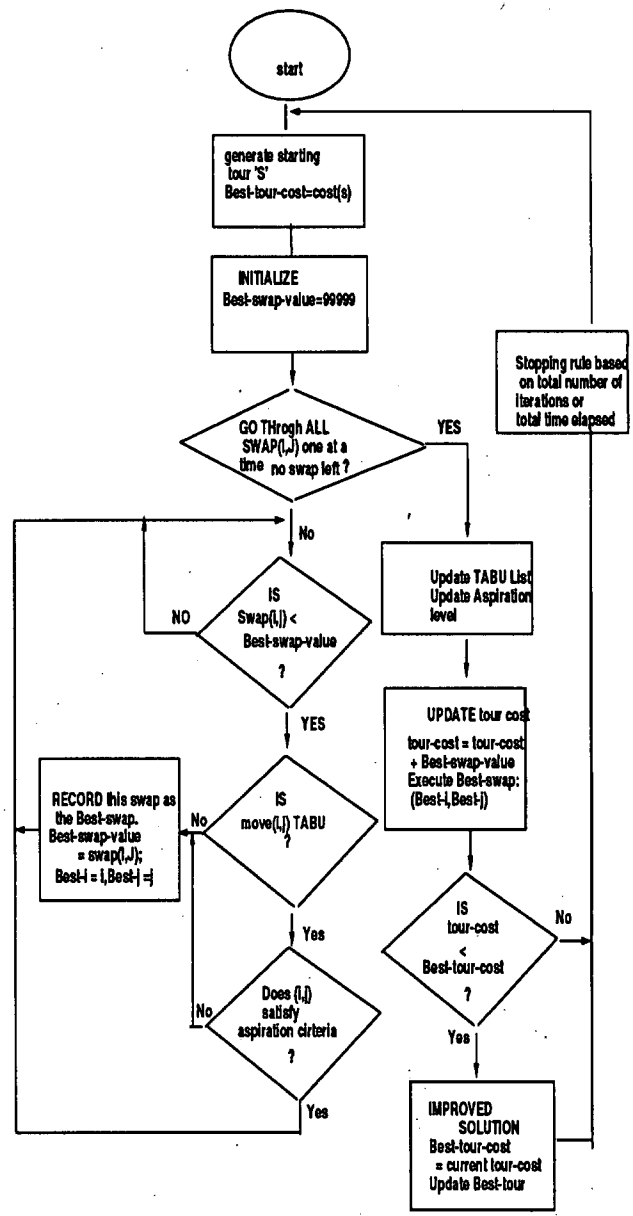


Figure 4: TABU Search

6. Vector(J, POSITION(J))
   to prevent city J from returning to POSITION(J).
7. City J
   to prevent city J from moving RIGHT of current position.
8. City J
   to prevent city J from moving in any direction.
9. Cities I and J
   to prevent both from moving.

Conditions 3 through 9 were established assuming that for cities I and J, POSITION(I) < POSITION(J). It is obvious that condition 1 is the least restrictive and 9 is the most restrictive. Conditions 3, 4 and 5 are also increasingly restrictive. Implementation issues are as follows.

- Data structures
  To determine the tabu status of a move and efficiently update the tabu list , we need well-designed data structures. As an example, tabu identification and tabu-list update for one of the tabu conditions (condition-4) are described below.

  We use two lists; Tabu-left and Tabu-list. Tabu-left indicates which cities are prevented from moving left of their current position. A tabu-list contains a fixed number of cities that had been moved to the right in the last k iterations (the Tabu-list size k is an input parameter). The Tabu-list is updated with a new city I which was moved right by incrementing the Tabu-list index (ring-index) and overwriting with city I at this new index position. This automatically removes the tabu status of the city which was at the position. For the index to stay within list range, increment is done using a mod operator: new-ring-index = (ring-index+1) mod tabu size. Similar data structures have been implemented for other tabu conditions.

- Aspiration criterion
  We used a simple aspiration criterion. Any tabu move is accepted that reduces the current tour length below the present best tour length. When the move results in a tour length lower than the best tour length, it indicates a tour not previously visited and so the move can no longer be considered tabu. This simple aspiration criterion is:

  $$TourLength + SwapValue(I, J) < BestTourLength$$

- Tabu list size
  This parameter must be tuned experimentally. For highly restrictive tabu conditions, the tabu list size must be smaller than those for less restrictive conditions. If the tabu list size is small, a cycling phenomenon will be evident, whereas, if it is large, the process might be driven away from the vicinity of global optimum. The optimum tabu list size will be the one which is long enough to prevent cycling but short enough to explore a continuum of solution space.

3.3 Problems of Local Search

Many problems may be encountered by local search routines. Due to the lack of global sampling capability, local search methods run the risk of being trapped in local minima. In fact, as the best solution for the 100-city TSP, SA obtained 21255 and TABU obtained 21352 under a variety of parameters, whereas the best known solution is 21247. It is obvious that these methods are trapped at local minima.

There are additional difficulties in parameter setting. Optimal parameters must be hand picked over a number of trials.

4 Hybrid GAs

One way to mitigate this problem is to combine GAs with local search (Figure 5). A hybrid algorithm that can combine the strengths of its component algorithms is expected to: (1) produce better solutions, (2) produce solutions with less computing cost, (3) automatically tune parameters, and (4) effectively handle larger problems (especially NP problems).

There are several ways to combine GAs with local search. This paper examines the combination of SA and TABU with GAs. GA+SA+TABU use both SA and TABU for local search and a GA is used as the global search manager.

Although it is possible to use a single local search mechanism, this runs the risk of being biased

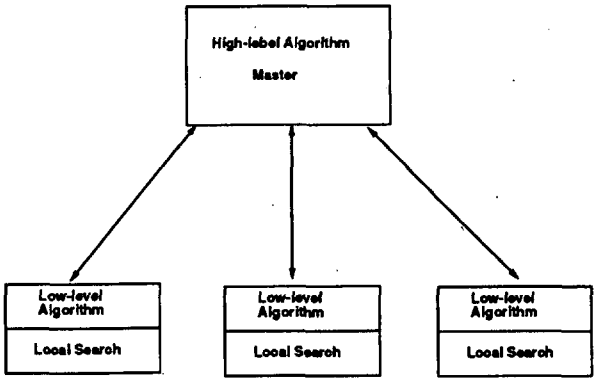


Figure 5: Organization of hybrid GA

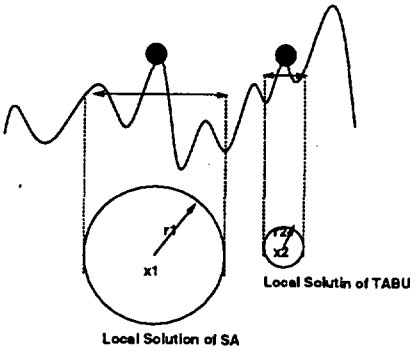


Figure 6: Scope of Local Search

Hybrid Algorithm using GA

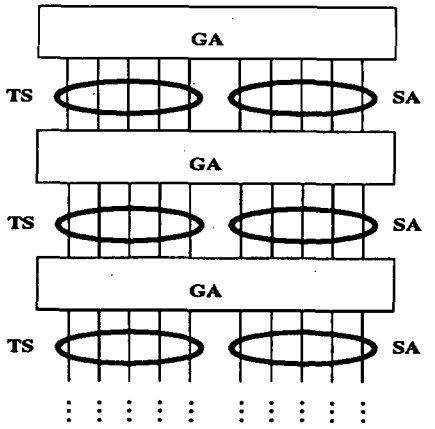


Figure 7: The idea of Hybrid GA

the characteristics of the local search mechanism (Figure 6).

For example, if we use a local search scheme with a rather larger local search area, the solution diversity would be minimized, undermining the utility of the GA. On the other hand, a local search mechanism with a small search area would place a heavy burden on the GA, because the GA requires sampling points close to the global optima.

By using multiple local search schemes, we expect to eliminate this problem. The basic idea of our method is to execute each low level algorithm (SA, TABU) for some specified amount of time, and leave result evaluation to the high level algorithm (GA) which restarts the low level routines in more promising areas in the solution space. This process is repeated as many times as necessary.

The overall algorithm for our method is as follows:

(1) Set N different initial tours. (2) Run TABU Search for Time T_t and get N_t local solutions. (3) Run SA for Time T_a and get N_a local solutions. (4) Get the new population with GA. (5) Return to 2 unless the stop condition has been met. This process is illustrated in Figure 7. In our experiment, $N = 100$ and $N_t = N_a = 50$. T_t and T_a should be selected heuristically. However, frequent solution exchanging among the local search routines seems to give good results. We used the elitist strategy for the GA, each time a new best-so-far solution has been found, an SA or TABU local search is initialized with that solution.

This algorithm can be realized with simulated annealing and tabu search implemented as subroutines. These subroutines could be executed sequentially followed by high level routine analysis. However, one of the most important features of this hybrid algorithm is the ease with which it may be executed in parallel. Each low level algorithm can be executed in parallel with a supervising process to synchronize execution and analyze results. This opens up the possibility of executing several low level algorithms in parallel, any number of which may be instances of SA or TABU with different operation parameters. Interprocess communication is minimal and only occurs between the high and low level algorithms. The algorithm can therefore be sped up linearly with the number of processors as long as they do not exceed the number of low level algorithms.

5 Experimental Results

The TSP experiments were performed on a Sun4/75 computer. Our programs were written in the C programming language.

Experiments were conducted using the 100-city problem, which has a known optimum solution of 21247 miles [4]. 100 tours were generated randomly and this set was used as the starting tour set by all algorithms (mean population size=100).

Table 1 shows the best solution, standard deviation, and computational cost for each method.

5.1 The Genetic Algorithm

The best solution found by the GA for the 100-city problem was 22253 miles. The average solution value was 23316 with a standard deviation 514 over 10 trials. The average time to find the best solution was about 640 seconds (290 generations). GA performance is sensitive to chromosome representation and the crossover operator used. We found that heuristic crossover (as described earlier) worked better than simple crossover or partial mapped crossover [2]. There has been an extensive comparative study of various crossover operators for the TSP [19].

5.2 Simulated Annealing

The simulated annealing algorithm has the following input parameters; the number of iterati-

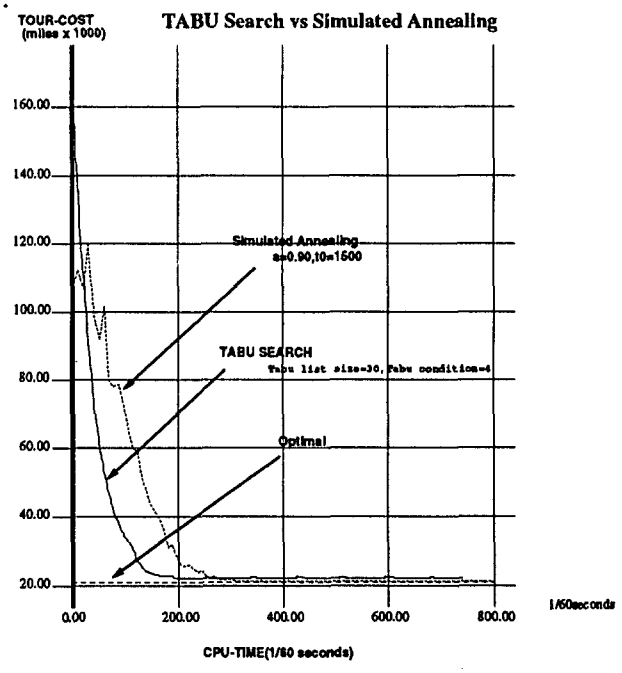


Figure 8: The best run by SA and TABU

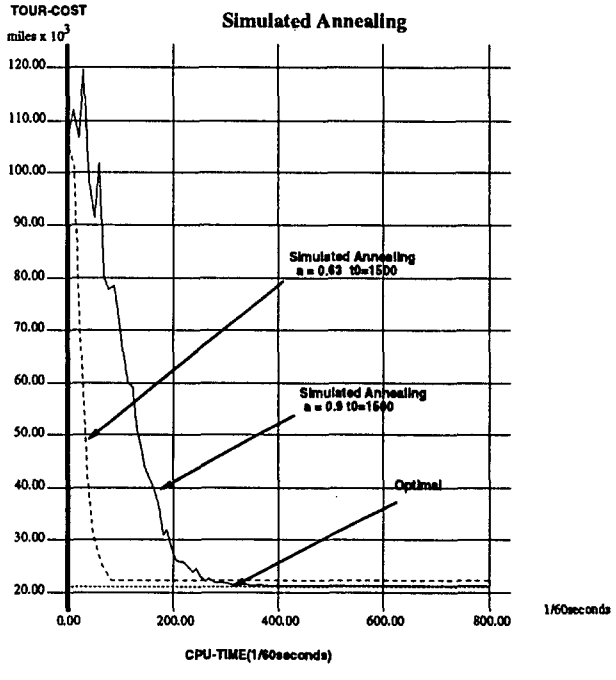


Figure 9: Effect of cooling schedule of SA

Algorithm	Best Cost	Average Cost	Standard Deviation	Average time(seconds)
GA	22253	23316	514	640 (290 generations)
TABU	21352	21433	68.4	210
SA($\alpha = 0.63$)	21331	21372	30.3	310
SA($\alpha = 0.90$)	21255	21284	27.5	1340
GA+SA+TABU	21247	21247	0	420 (5 generations)

Table 1: Experimental Results

ons to approximate equilibrium, starting temperature, and cooling rate α. They allow the algorithm to be tuned for specific problems. SA performance is very sensitive to the cooling schedule. In Figure 9, we show how cooling schedule affects SA. The best solution found by SA($\alpha = 0.63$) for the 100-city problem was 21331 miles. The average solution was 21372 with a standard deviation of the solutions over 10 trials of 30.3. Average time to best solution is about 310.

The best solution found by SA($\alpha = 0.90$) was 21255 miles. The average solution was 21284 with a standard deviation 27.5 over 10 trials. The average time to find the best solution was about 1340 seconds.

The convergence curve is shown in Figure 8. In this case, we never reached the optimal solution.

5.3 Tabu search

The first stage in developing the tabu search algorithm was to implement the hill climbing heuristic (In this case 2-opt). We then transformed this into our tabu search routine using the nine different tabu conditions discussed earlier. The tabu search process has the following input parameters: tabu condition, tabu list size, and total number of iterations. The tabu condition and list size are interdependent parameters. The algorithm is very sensitive to them. A small tabu list size with a weak tabu condition results in cycling, whereas, a large list with a strong tabu condition drives search away from the global optimum. To reach a compromise, we conducted experiments for each of the nine tabu conditions to find reasonable tabu list sizes. Generally, the list sizes range from a quarter to a third of the number of cities for condition 4 and 7. A size of about a fifth for conditions 5, 8, and 9 gave the best results for the problem tested. Conditions 1, 2, 3, and 6 required tabu list sizes in the vicinity of the problem size, i.e., the number of cities. On average, tabu conditions 4 and 7 produced better results in less time than other conditions. In our experiment, the best solution found by TABU for the 100-city problem was 21352 miles. The average solution was 21433 with a standard deviation 68.4 over 10 trials. The average time to best solution was about 210 seconds. Examining the problem over 1000 trials, we never reached the optimal solution. The convergence curve is shown in Figure 8.

5.4 GA+SA+TABU

The convergence curve is shown for this method in Figure 10. The result clearly shows the importance in combining multiple local search methods with the GA.

5.5 GA+SA and GA+TABU

We examined the convergence of GA+SA, GA+TABU, and GA+SA+TABU(Figure 11). GA+SA finds better solutions than SA. Also, GA+TABU finds better solutions than TABU. These results show that the GA is useful as a high level algorithm which controls low lovel algorithms. However, the result of GA+SA+TABU produced better results than GA+SA or GA+TABU. In our 100-city TSP, GA+SA+TABU always found better solutions in fewer generations.

6 Discussion

For the four search algorithms we investigated, our experiments demonstrated the following;

Simulated Annealing: If the cooling schedule is very carefully determined, SA can find near

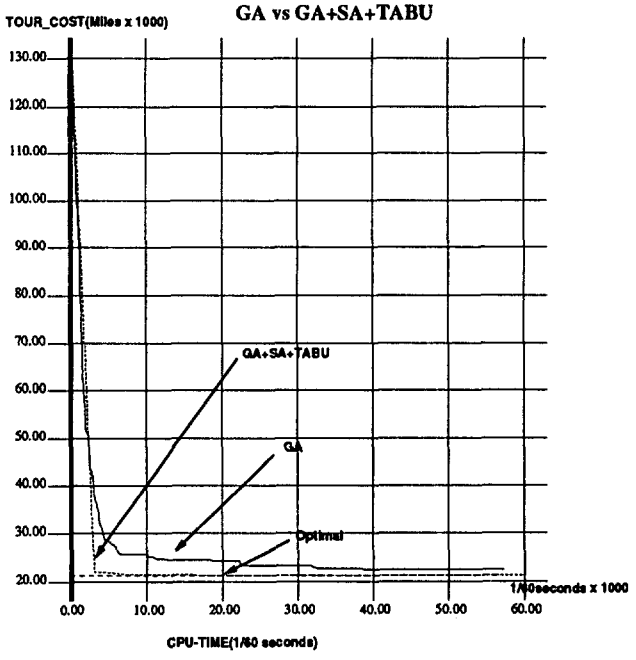


Figure 10: The best run for GA and GA+TABU+SA

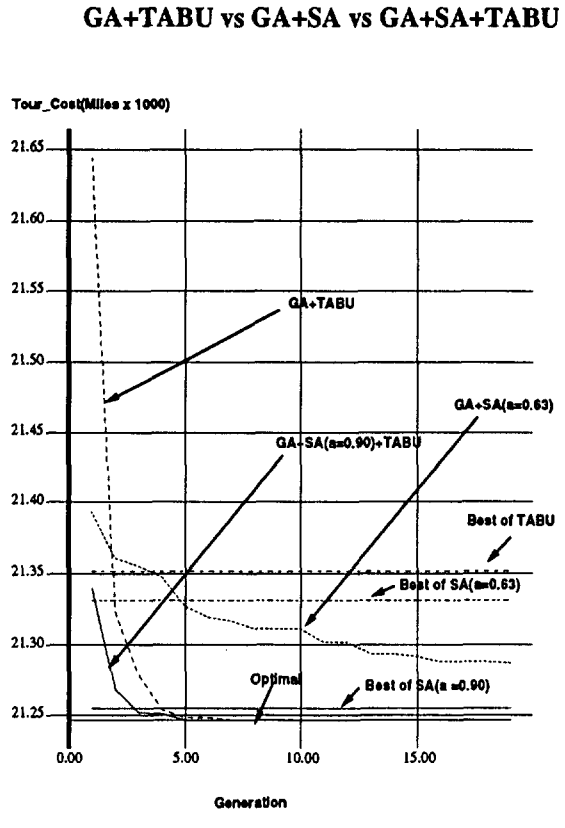


Figure 11: Performance Comparison

optimal solutions better than GA or TABU. Standard deviation of local solutions from SA is smaller than that from TABU. The problem with SA is in tuning the cooling schedule,which is a computationally expensive task.

TABU Search: TABU can find local solutions faster than GA or SA, but the quality of solution is not as good as that of SA. Standard deviation of local solutions from TABU is larger than that from SA. The tabu conditions must be selected heuristically since they depend on the problem.

Genetic Algorithms: GA performance is sensitive to chromosome representation and the crossover operator used. Better representation and crossover and local search combination methods are needed to improve performance.

Hybrid GA: A GA itself does not have local search capabilities, but it can be improved by combination with a local search algorithm. Our hybrid GA has better solution quality than the GA or any of the local search algorithms investigated. Our experiments showed that GA+SA+TABU always found optimal solution in the 100-city TSP and is superior to SA, TABU, GA+SA, or GA+TABU. We found that solution recombination (i.e., crossover) is very useful for finding the optimal solution. We also found that using multiple local search algorithms is very useful for improving performance.

7 Conclusions and Future Work

In this paper, we proposed and examined a hybrid GA which uses more than one local search routines with a GA. For the local routines, we used TABU and SA. Experimental results were presented for a 100-city TSP. These results demonstrate that this approach to combine TABU and SA as local searchers for the GA is very promising. It seems that a good hybrid method should use at least two different local search methods. We do not think, however, that this method is best one for TSP. Other combinations are possible, such as using Lin-Kernighan (LK) algorithm and GAs.

As Malek described, hybrid algorithms are very well suited for a variety of problems. With the advent of parallel machines, inherent parallelism in hybrid algorithms becomes especially attractive. Yet another advantage of proposed approach is its ability to tolerate software faults due to multiple algorithm implementations.

8 Acknowledgments

We have benefited immensely from the many suggestions and discussions with Miroslaw Malek and Hiroaki Kitano.

References

[1] E. L. Lawler et al. *The Traveling Salesman Problem, A Guided Tour of Combinatorial Optimization,* Wiley, 1990.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning,* Addison-Wesley, Reading, Mass, 1989.

[3] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi, *Optimization by Simulated Annealing,* Science volume 220, pp. 671–680, 1983.

[4] Miroslaw Malek, Mohan Guruswamy, et al.,: *A Hybrid ALGORITHM TECHNIQUE,* technical report of Texas Univ at Austin, TR-89-06, 1990.

[5] Hiroaki Kitano, *Empirical Studies on the Speed of Convergence of Neural Network Training using Genetic Algorithms,* AAAI-90 Proceeding Eight National Conference on Artificial Intelligence, 1990.

[6] Lawrence Davis, *Handbook of Genetic Algorithms* , Van Nostrand Reinhold, 1991.

[7] *Genetic Algorithms and Simulated Annealing*, Lawrence Davis (ed): Pitman, London, Morgan Kaufmann Publishers Inc., Los Altos, California, 1987.

[8] Powell, D., Tong, S. and Skolnick, M.: *EnGENEous: Domain independent, machine learning for design optimization,* Proc.of ICGA-89, 1989.

[9] S. Geman, D. Geman: *Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images,* IEEE Trans. Patt. Anan. Mac. Int. 6, pp. 721-741, 1984.

[10] H. Szu, R. Hartley: *FAST SIMULATED ANNEALING,* Phys. Lett. A 122, pp. 157-162, 1987.

[11] John J. Grefenstette, Rajeev Gopal, Brian Rosmatia, Dirk Van Gucht: *Genetic Algorithms for the Traveling Salesman Problem,* Proceeding of 1st International Conference on Genetic Algorithms and Their Applications, John J. Grefenstette (ed), 1985.

[12] Padberg, M. and Rinaldi G: *Optimization of a 532-city symmetric traveling salesman problem by branch and cut,* Operations Res. Lett. 6, pp. 224 - 230, 1987.

[13] Johnson, S. D.: *Local Optimization and the Traveling Salesman Problem,* Proceedings of the 17th Colloquium on Automata, Languadges and Programming, Springer-Verlag, pp. 446-461, 1990.

[14] TABU search-part I. ORSA Journal on Computing 1(3) pp. 190-206.

[15] TABU search-part II. ORSA Journal on Computing 2(1) pp. 4-32.

[16] Muhlenbein, H.: Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization, in the J. D. Schaffer (ed), Proceeding of The Third International Conference for Genetic Algorithms, Morgan Kaufmann Publishers, Inc., pp. 224 - 230.

[17] Lister, R.: Annealing Networks and Fractal Landscapes, IEEE International Conference on Neural Networks, San Francisco, March 1993, Vol. I, pp. 257 - 262.

[18] X. Yao.: Optimization by genetic annealing, Proc. of Second Australian Conf. on Neural Networks, 1991, pp. 94 - 97.

[19] X. Yao.: An empirical study of genetic operators in genetic algorithms, Microprogramming and Microprocessing, 38 (1 - 5), pp. 707 - 714, 1993.

Appendix

A 532-city TSP

We challenged our method on a larger problem: a 532-city TSP. Our 532 city TSP experiments were performed on a Sun4 (Sparc-station-10) computer. First we randomly initialized the tour. This approach proved computationally expensive. Next, we initialized the tour by the nearest neighbor method. Figures 12 and refinit-neighbour show the initial tours generated at random and by the nearest neighbour method. The published optimal solution for the 532-city problem is 27686 by Padberg. This corresponds to a tour cost of 86900 miles in our experiment. Our best result after 30 generations was 88415. Figure 14 shows the tour generated by GA+SA+TABU after 30 generations, which was reached at 18th generation. In this case, the relative deviation from optimal tour length is 0.0174. This result is comparable to the result in Muhlenbein [16].

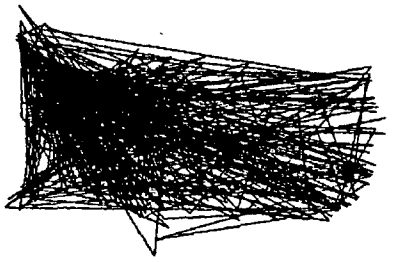


Figure 12: Randomly generated initial tour

Figure 13: Initial tour by the nearest neighbor method

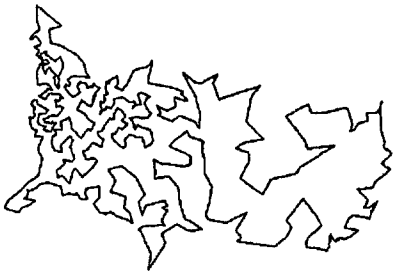


Figure 14: Best tour by GA+SA+TABU (30 generations)

Sacrificial Acts in Single Round Prisoner's Dilemma

Masaru Tomita
Department of Enviromental Information, Keio University
5322, Endo Fujisawa, 252 Japan
mt@sfc.keio.ac.jp
AND
Takashi Kido
Faculty of Science and Technology, Keio University
3-14-1, Hiyoshi, Kohoku-ku, Yokohama, 223, Japan
kido@math.keio.ac.jp

Keywords: prisoner's dilemma, suicide acts

Edited by: Xin Yao
Received: November 16, 1993 **Revised:** April 11, 1994 **Accepted:** April 18, 1994

In this paper, we say an act is sacrificial if it hurts the actor as an individual but benefits his species as a whole. We will present two situations in single round prisoner's dilemma where such sacrificial acts are so important that species cannot survive without them. Those two cases are: Cooperate with Justice (CWJ) and Cooperate with Suicide (CWS). The former is an act to punish defectors. The latter is more counter-intuituve; an act to hurt himself for nothing. We will show by computer simulation that those two strategies survive in population dynamics, while the simple Cooperate (C) strategies cannot survive in the same situations.

1 Introduction

The game of Prisoner's Dilemma has been studied for many years [1, 2, 3, 5, 6, 7, 8, 9, 10, 11]. It is a two player non-zero-sum game, and each player has to choose from two actions: Cooperate (C) and Defect (D). Its pay-off matrix is shown below.

	C	D
C	3/3	0/5
D	5/0	1/1

Iterative Prisoner's Dilemma is that a player plays Prisoner's Dilemma multiple times in a row with the same opponent. Axelrod organized tournaments of Iterative Prisoner's Dilemma in 1981 and 1988 [1, 2, 3]. The most successful strategy in the tournaments is known as TIT-FOR-TAT. This strategy is to cooperate as long as the opponent cooprates, and to retaliate by defecting in the next round if the opponent defected. This retaliation will discourage the opponent from defecting, and higher pay-off can be expected throughout the rest of the rounds. It is important to note that TIT-FOR-TAT is effective because a player will play many rounds with the same opponent.

2 Single Round Prisoner's Dilemma

In the single round Prisoner's Dilemma, on the other hand, the strategy of TIT-FOR-TAT cannot be used. If you are defected, sorry, that is the last (and first) game with the opponent, and there is no opportunity to retaliate. If you still defect in the next round for retaliation as in TIT-FOR-TAT, you would retaliate a different opponent, who may be innocent from defecting in the previous round. Thus, in the Single Round Prisner's Dilemma, the most beneficial strategy is to always defect. As the pay-off matrix indicates, you will always get higher pay-off no matter what action the opponent will take.

Let us now consider population dynamics of two species: Cooperate (C) and Defect (D). Each individual of the Cooperate species always coope-

rates, and each individual of the Defect species always defects. We then play a round-robin tournament of Single Round Prisner's Dilemma, and the survivability of each individual is proportional to the total pay-off value he has obtained. We also assume that the size of total population is constant at each generation.

More precisely, let $S_1 ... S_n$ be n different species (strategies) in the population, where $n = 2$, $S_1 = C$, and $S_2 = D$.
Let $P_{i,k}$ be the size of population of S_i over the entire population in the k-th generation. Thus, $\sum_{i=1}^{n} P_{i,k} = 1$ for all k.
Let $E(S_i, S_j)$ be a pay-off function which returns the pay-off value of S_i against S_j.

Now the dynamics can be defined as follows:

$$P_{m,k+1} = \frac{\sum_j E(S_m, S_j) \cdot P_{m,k} \cdot P_{j,k}}{\sum_{i,j} E(S_i, S_j) \cdot P_{i,k} \cdot P_{j,k}}$$

Figure 1 is the result of simulation with the initial population ratio of 0.5 and 0.5 ($P_{1,0} = P_{2,0} = 0.5$). It takes only several generations for C's to be beaten by D's and wiped out from the population.

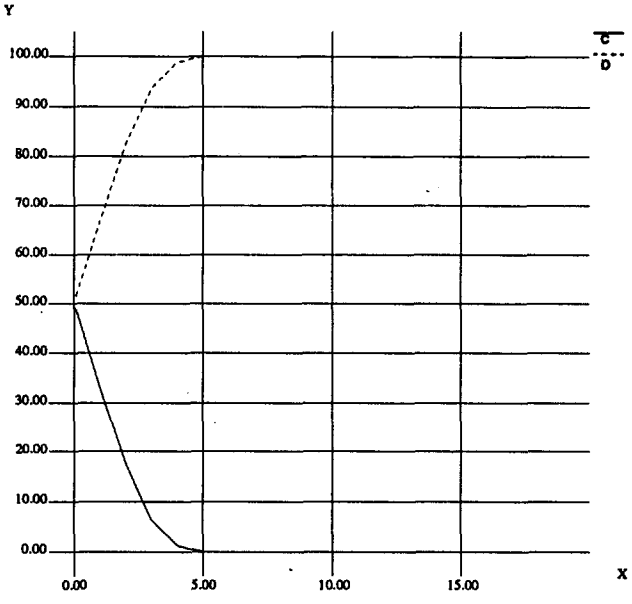


Figure 1: C and D, historical diagram

3 Sacrificial Acts: Cooperate with Justice

Suppose that a player has a chance to punish defecters in the Single Round Prisner's Dilemma. If a player is defected, he has an option to exercise the "act of justice", by forfeiting all of the defecter's five (5) points gained by the defective action. Unlike TIT-FOR-TAT, the "act of justice" is expensive; it costs one (1) point to exercise the act.

Let us consider a new species, "cooperate with justice" (CWJ). CWJ behaves like C against all of the species including CWJ itself. However, after CWJ gets defected by a D, it will execute the "act of justice", to make sure that D's defective action is never rewarded.

Notice that the act of justice by a CWJ is sacrificial. [1] CWJ has to pay one point, and due to single round mat, it cannot benefit from the opponent's penitential behavior (if any) for the rest of the game. Thus, in terms of individual pay-off, this act does not make sense; a CWJ only hurts himself. However, the sacrificial act by an individual CWJ can greatly benefit the entire CWJ's, because it damages their natural enemies, i.e., Defecters.

The payoff matrix with CWJ instead of C is shown below.

	CWJ	D
CWJ	3/3	-1/0
D	0/-1	1/1

The result of simulation with the same initial population ratio (0.5 and 0.5) is shown in figure 2. As we can see in the figure, D's get wiped out by the acts of justice, and CWJ's will soon dominate the entire population. We can therefore conclude that we have shown a clear case where individual's sacrificial acts can help the species as a whole: C's without the sacrificial acts get ruined by D's, but C's with the sacrificial acts (CWJ's) ruin D's.

[1] TIT-FOR-TAT, on the other hand, can be considered as "selfish" action, because (1) payoff of the particular defective action is greater than a cooperative action, and (2) it expects the opponent to refrain from defecting, increasing its total payoff throughout the rest of the game.

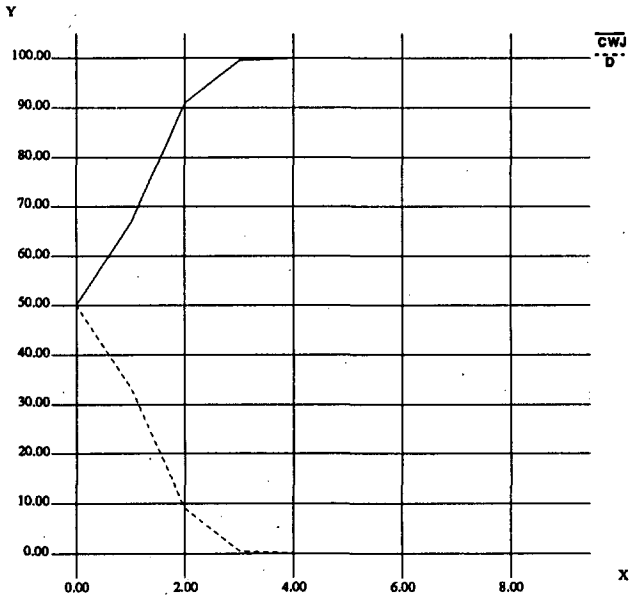


Figure 2: CWJ and D, historical diagram

4 Free Riders: Cooperate without Justice

In this section, we shall consider population of three species: Defectors (D), Cooperators with Justice (CWJ) and Cooperators without Justice (C). The pay-off matrix is shown below.

	C	CWJ	D
C	3/3	3/3	0/5
CWJ	3/3	3/3	-1/0
D	5/0	0/-1	1/1

We did an experiment similar to the previous sections with

$$S_1 = C, S_2 = CWJ, S_3 = D, P_{1,0} = 0.2, P_{2,0} = 0.45, P_{3,0} = 0.35$$

The result is shown in figure 3. At the first 20 generations or so, D's are largely controlled by CWJ's acts of justice. C's and CWJ's are increasing at the beginning, because decrease of D's benefits both C's and CWJ's. However, only CWJ's are paying the price for controlling D's, and thus, C's are "free riders". This makes C's predominant over CWJ's, and the population of CWJ's gradually decreases due to the existence of C's. And at the 17th generation, the population of CWJ's gets so small that they can no longer control D's. D's then revive, increasing the size of their population, and eventually, D's dominate the entire population as in section 2.

In short, free riders (C) unwilling to pay the price will hurt their best friends (CWJ) and in turn they (C) will get ruined by enemies (D).

This result is consistent with the result of Axelrod's work [1]. In his paper, Axelrod suggested the notion of "meta-norm"; that is, one should punish not only defectors but also those who do not punish defectors.

In the next section, however, we introduce a different notion, "suicide", that can help stabilize population of the species.

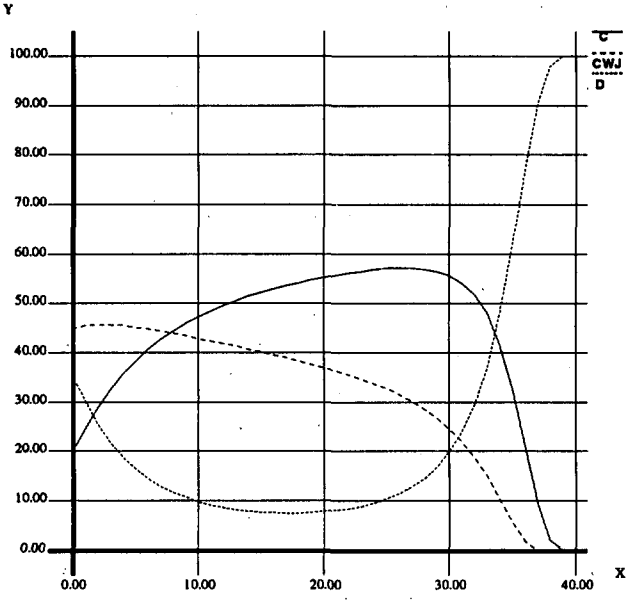


Figure 3: C and CWJ and D, historical diagram

5 Suicidal Acts: Cooperate with Suicide

In the previous section, we described sacrificial acts. That is, reducing points of his own in order to reduce significantly more points from enemies of his species. Thus, it is intuitively sound that this kind of sacrificial acts can help his species as a whole.

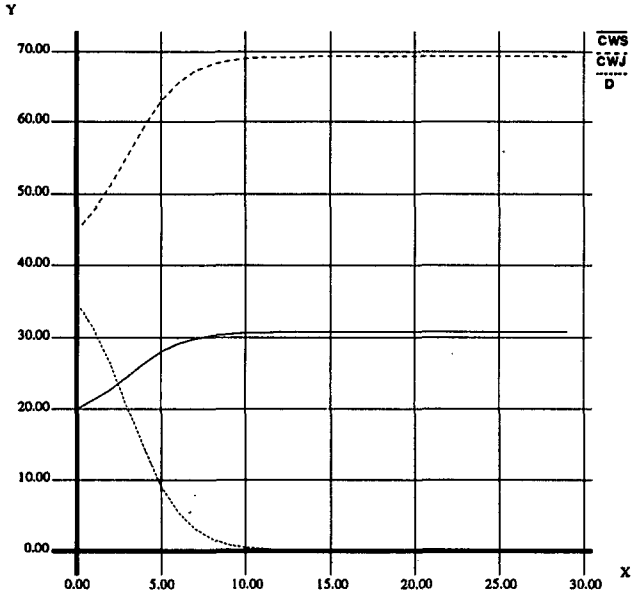


Figure 4: CWS and CWJ and D, historical diagram

In this section, we introduce less intuitive acts of "suicide". The act of suicide reduces points of his own, and does not reduce or increase his opponet's point. It simply hurts oneself without any return. This seemingly counter-intuitive act can help stabilize the population of the species, as demonstrated in the following experiment.

In the experiment, we have D's, CWJ's, and CWS's (Cooperate with Suicide) instead of C's in the previous experiment. The pay-off matrix is shown below.

	CWS	CWJ	D
CWS	3/3	3/3	-1/5
CWJ	3/3	3/3	-1/0
D	5/0	0/-1	1/1

Only the difference between this matrix and the previous matrix is C and CWS. That is, if a CWS is defected by a D, rather than do nothing (like C's in the previous experiment), he commits suicide reducing one point of his own. The opponent will keep the 5 points gained by the defective action.

The result of the experiment with the same initial population ratio ($P_{1,0} = 0.2, P_{2,0} = 0.45, P_{3,0} = 0.35$) is shown in figure 4. In the short term, the suicidal acts by CWS's hurt themselves, resulting less rapid growth than in figure 3. However, in the long term, the suicidal acts help CWJ's survive; CWS's slow growth keeps CWJ's from being predominated, and D's (CWS's enemies) can be kept controlled by CWJ's.

6 General Analysis

The process at work can be visualized with more generality and precision if we shift from the historical illustration to the phase diagram in figure 5 [4]. In such a diagram any point of the triangle represents a distribution of the population over the three stragegy options. The vertical coordinate indicates the proportion of "CWJ", the horizontal coordinate shows the proportion of "C", and finally the remaining proportion of "D" is measured by the horizontal distance from the point to the hypotenuse. The population dyanamics is effected by the initial population ratios as illustrated in Figure 5. There are two kinds of evolutionary equilibrium (EE). (1) EE is at Vertex D. (2) EE is along Edge C and CWJ. Figure 6 shows this situation. If we select the initial population ratio from the black part of the triangle, D exploits C and CWJ. And If we select the initial population ratio from the white part of the triangle, D becomes extinct, C and CWJ can survive. In the case of CWS and CWJ and D, the results are shown in Figure 7. The black part of the Figure 6 are a little larger than that of Figure 7. Figure 8 shows the difference between Fingre 7 and Figure 8 where suicide act can be meaningful.

7 Concluding Remarks

In this paper, we have presented a situation where species with sacrificial acts can survive while those without would not. We have also presented a situation where species with suicide can survive while those without would not.

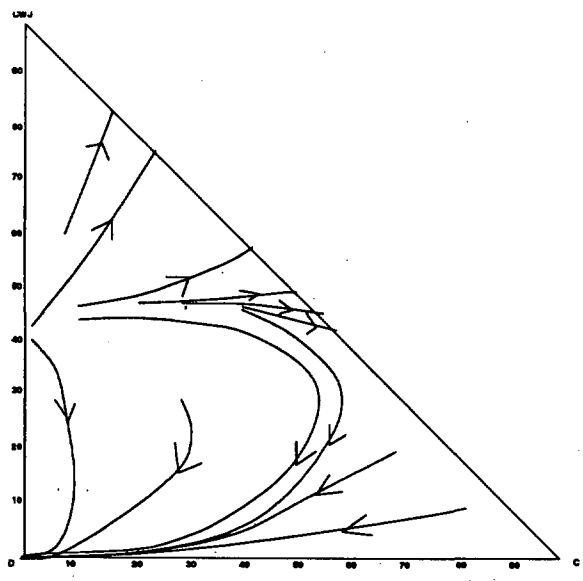


Figure 5: C and CWJ and D, phase diagram

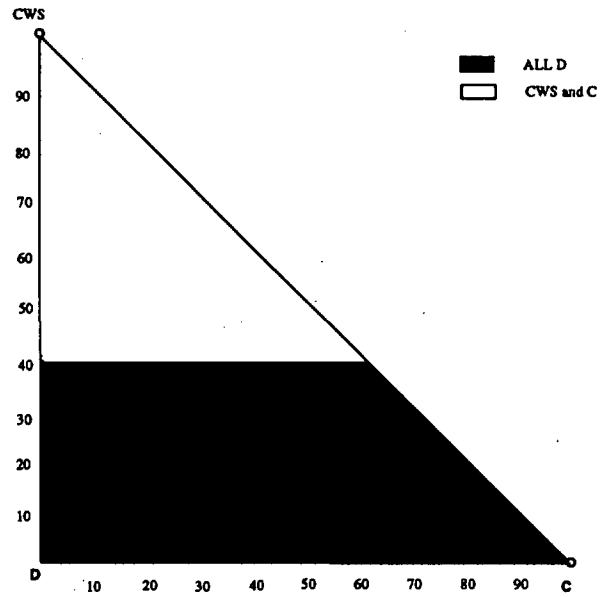


Figure 7: CWS and CWJ and D, evolutionary equilibrium

Figure 5: C and CWJ and D , phase diagram

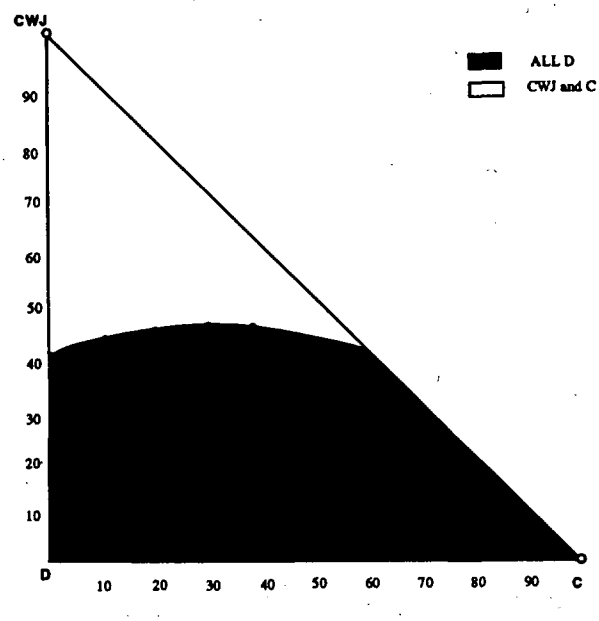


Figure 6: C and CWJ and D, evolutionary equilibrium

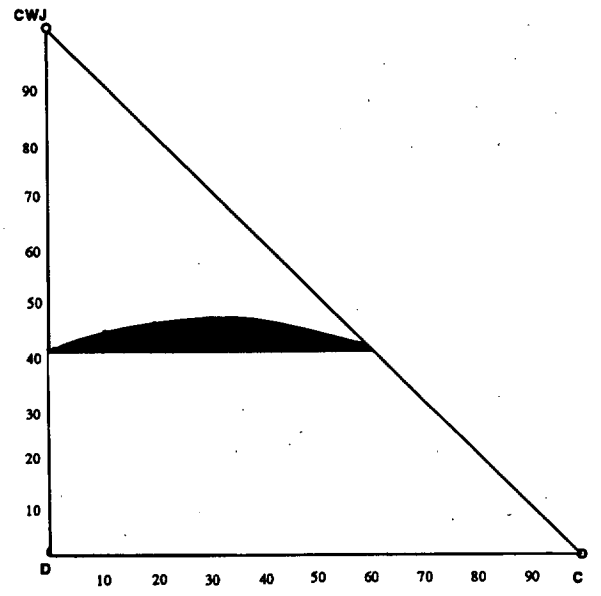


Figure 8: The regions that suicide act can be meaningful

References

[1] R.Axelrod: The Evolution of Cooperation, Basic Books, 1984.

[2] R.Axelrod, W.D.Hamilton: The Evolution of Cooperation, Science, vol.211, 27 March 1981, pp.1390-1396.

[3] R.Axelrod, D.Dion: The Further Evolution of Cooperation, Science, vol. 242, 9 Dec 1988, pp.1385-1390.

[4] J.Maynard Smith: Evolution and The Theory of Games, Cambridge Univ. Press, 1982.

[5] Cory Fujiki: Using the Genetic Algorithm to generate LISP source code to solve the Prisoner's Dilemma, ICGA 2nd, 1987, pp.236-240.

[6] James P. Kahan, Dwight J. Goehring: The Uniform N-Person Prisoner's Dilemma Game, Journal of Conflict Resolution, vol.20(1), Mar 1976, pp.111-128.

[7] Marks R.E.: Breeding Hybrid Strategies, ICGA 3rd, 1989, pp.198-207.

[8] John R.Hauser, Peter S. Fader: Implicit Coalitions in a Generalized Prisoner's Dilemma, Journal of Conflict Resolution, vol. 32(2), Jun 1988, pp.402-408.

[9] Theodore To: More Realism in the Prisoner's Dilemma, Journal of Conflict Resolution, vol:32(2), Jun 1988, pp.402-408.

[10] K.Lindgren: Evolutionary Phenomena in Simple Dynamics, Artificial Life II, Addison-Wesley, 1992.

[11] Jack Hirshleifer, J.C.Martinez Coll: What Strategies Can Support the Evolutionary Emergence of Cooperation?, Journal of Conflict Resolution, vol.32(2), Jun 1988, pp.367-398.

[12] David B. Fogel: Evolving Behaviors in the Iterated Prisoner's Dilemma, journal of evolutionary computation, 1993, pp77 – pp97.

From Evolutionary Computation to Computational Evolution

Jari Vaario
Evolutionary Systems Department, ATR Human Information Processing Research Laboratories
2-2 Hikari-dai, Seika-cho, Soraku-gun, Kyoto 619-02, JAPAN
Phone: +81 7749 5 1004, Fax: +81 7749 5 1008
jari@hip.atr.co.jp

Keywords: evolutionary computation, growing neural networks, adaptive behavior

Edited by: Xin Yao
Received: November 16, 1993 **Revised:** April 11, 1994 **Accepted:** April 18, 1994

This paper proposes that evolutionary computation be understood more like computational evolution, i.e., *to use the evolutionary process for* construction, *rather than for optimization. For this purpose simulation of the evolutionary process should include a non-linear developmental process from genotype to phenotype. In this developmental process the environment has an important role. In order to model the developmental process under the influence of the environment, a new modeling language is introduced. The focus of this language is on the interactions, which are considered to be the basic elements for environmental adaptation. The developed modeling method provides a complete simulation environment for the construction of organisms. The developed system aims to construct intelligence as adaptive behavior based on artificial neural networks.*

1 Introduction

Artificial Intelligence has failed to model intelligence as it appears in natural systems. This is an actual problem for applications that should behave independently in an unknown and unpredictable environment, *e.g.*, *intelligent autonomous systems*.

There are two points where natural systems depart remarkably from common engineering principles. First, information from the old system to the new system is transformed in two parts: the *genetic information* and the *cell machine* needed to interpret the former. The genetic information does not directly define the final result, but instead describes it in a non-linear way through the development process.

Second, the development process that leads from a single cell to a multicellular organism is a non-linear process that happens in close interaction with the environment. For the final result the environment is as important as the initial description. The systems that are capable of *self-modification* in the dynamic environment will be favored in evolution.

This approach has implications for engineering: instead of direct design, the complex systems are engineered indirectly. The focus will be on self-adaptation rather than on the final functionality. The design subject may be, in principle, anything, but here we will restrict our analysis to autonomous systems, and in particular to control unit based on simple cellular structures.

2 Background

In engineering the usual concept is to separate a system from the environment by replacing it with some system parameters. This is based on the old understanding that biological organisms are well bounded systems. However, as engineers – like biologists – become increasingly aware of the embodiment of the system in the environment, they should concentrate their efforts on finding a mechanism to lower the system bounding conditions, which will lead toward the design of *general* adaptive systems.

In the following we review how the understanding of biological systems has been changed to-

ward more tightly embodied systems, and then, following by this, we introduce the concept of computational evolution in contrast to evolutionary computation.

2.1 Environmental Embodiment

Opinions of how much the environment effects the development and evolution of natural systems are wide-ranging. According to the neo-Darwinian view, the effect of the environment applies only to the natural selection process.

This opinion is challenged by Piagetian environmental adaptation, where the actions of organisms cause a continuous feedback from the environment, and behavior is considered to be the prime source of evolution. [1, 6]. The next step has been proposed by Wesson [13] with arguments of direct genetic adaptations to the environmental changes. Both of these opinions suggest that the Darwinian natural selection mechanism is inadequate for creating the biological life now present in nature. This is the same conclusion than Stuart Kauffman made in [2]. The environment has to have a more direct method for affecting the direction of evolution than Darwinian survival selection.

These theories can be compared to software technology. Current software technologies are based on the idea of predetermination, *i.e.*, the program does not change after it has been written. Recently the Darwian ideas of selectionism have been applied by Koza [3] as Genetic Programming. The next step is to extend software according to the general theories of self-organizing systems.

2.2 Structuralism vs. Functionalism

The recent experiments in modeling the evolutionary process have once again raised the old question of the actual relation between form and function. The previous topic was wether "structuralism" (the primacy of form over function) or "functionalism" (the primacy of function over form) is more valid (see, for example, [8]. Now the same question seems to be emerging from evolutionary computation: Do we need to model the form in order to create a function, or could we concentrate only on the function and forget the form in the simulations of evolution?

Current models of evolutionary computation ignore the form and concentrate on the explicitly defined functions. This approach leads to a powerful "search" mechanism in a known function space, ie., optimization, but does not demonstrate any of the creativity that is clearly visible in natural systems. For modeling this creativeness, the form, and its developmental process, must also be included in the models.

2.3 Computational Evolution

We would like to make a distinction between the research on genetic algorithms, *i.e.*, evolutionary computation, and our research. We would like to explore purposeless evolution, not optimization in a known space. However, purposeless is not meaningless. We have a mechanism to create structures that result in behavior in the environment. The explicitly defined fitness function is replaced by an implicitly defined behavior capable of reproduction. We call this *Computational Evolution*, rather than evolutionary computation. The difference between these two approaches can be seen in Figure 1.

The idea is to find the mechanism by which the biological systems respond to novel problems. This could also be called adaptive behavior. The question is does there exist any single mechanism, or principle, that gives rise to organisms and their "intelligent" behavior. If there exists such a mechanism, could it be transferred to artificial systems to provide the principle for spontaneous intelligent behavior.

It is clear that evolution (phylogenesis) based only on a selection mechanism does not provide an answer. It must be combined with morphogenesis and ontogenesis. How to make a computational model for all these processes is the problem that must first be answered. After we have a computational model for these processes, we can explore the effect of each of them in order to create artificial systems.

2.4 Focus of this research

The above theoretical arguments should be elaborated with concrete examples. A computational model would be very useful to verify what kind of environmental effects can be modeled, and how these effects can replace genetic predetermination.

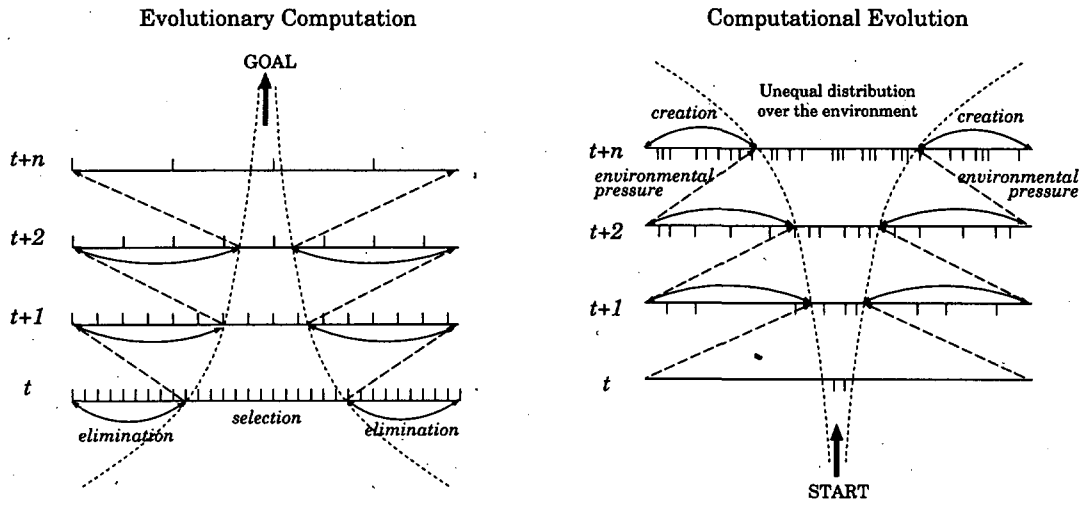


Figure 1: Comparison between Evolutionary Computation and Computational Evolution. The result computational evolution is an unequal distribution of individuals. The creation of new individuals will fill the space and exert a pressure on the existing individuals. In evolutionary computation this pressure is replaced by the explicitly defined selection of the fittest individuals. This explicitness destroys the creativity of evolution.

A computational model is capable of capturing the essence of the above arguments, and makes it is easy to conduct experiments with different initial conditions.

However, it is not obvious what kind of computational model is most suitable. First, we have to recognize that the model should be capable of embedding organisms in the environment. Second, the model should be capable of the *local* interactions between the organisms and the environment. Third, the model should be capable of covering all the different environmental adaptation phases. These include the development of an organism (morphogenesis), the plasticity that results in behavior (ontogenesis), and a selection mechanism with genetic variations (phylogenesis). These processes are illustrated in Figure 2.

We have thus far reported the results of a simulation mechanism as a method to construct intelligent behavior in [12, 10]. A mechanism for evolutionary design and model building has been discussed in [11, 5]. A detailed description of the simulation mechanism can be found in [9]. This paper is a continuation of the previous work reporting the latest results.

3 Computational Model

A computational model is based on the concept of production rules inspired by Lindenmayer systems [7]. However, instead of using a linear string of letters we describe abstract objects, which have their own production rules to execute.

Unlike L-systems having position ordered parameters for letters, we use attributes (a tuple of key and value) to describe the parameters of objects. Each object can also have sub-objects with their own production rules and attributes. This forms a hierarchical representation of the objects that also allow an environment model in the same model.

Because of these fundamental changes we prefer to call our system Multilevel Interaction Simulation language (MLIS) [9].

3.1 Multilevel Interaction Simulation language

In traditional object-oriented systems the action of an object is based on the sent message, *i.e.*, the object is passive unless someone sends a message for action. This implies that the message control mechanism must be synchronized in the sense that the sender must know to whom and

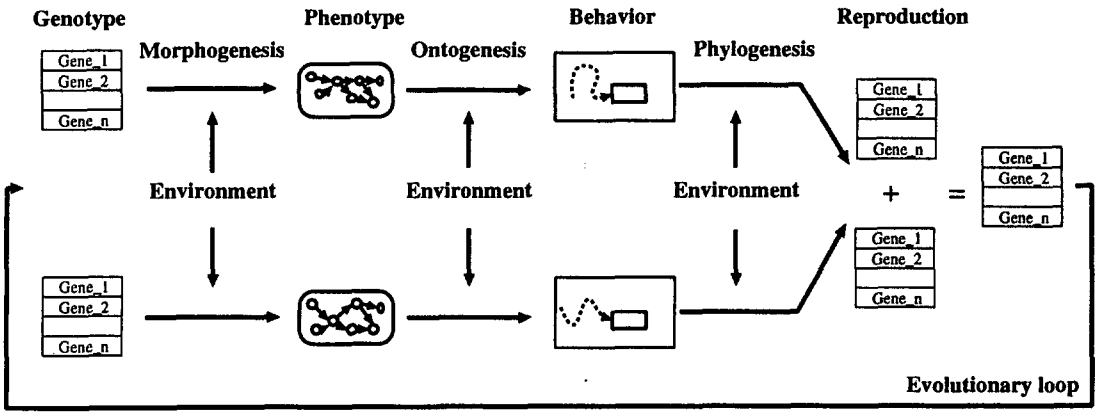


Figure 2: Possible environmental adaptations: morphogenesis, ontogenesis and phylogenesis. The organism starts with genetic information and a basic mechanism to interpret it, and interact with the environment (cell) resulting in a phenotype, and further behavior. Through a selection process based on the result of the earlier phases, new genetic information with an interpretation machine (cell) results in closing the evolutionary loop.

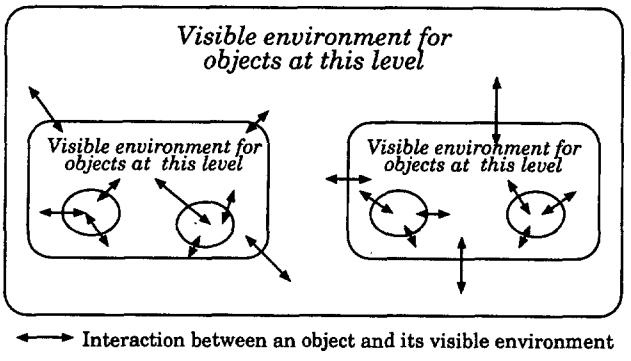


Figure 3: The hierarchical organization of objects. Each object has its own local visible environment. The interactions are executed in parallel and simultaneously.

when to send the messages.

In the MLIS language each object is actively "checking" the environment and, based on the information in the environment, the objects take appropriate actions. The environment visible to each object depends on its position in the hierarchical organization (see Figure 3). Because each object is autonomously executing its own instructions, the system possesses a strong parallelism.

The simulation of such a system must focus on the interactions (actions that the objects take based on the visible environment). Each object should possess a set of definitions describing the actions to be taken, *i.e.*, a set of production rules. The general form of these rules is $f(x+1) = g(f(x), e(x))$, where $f(x)$ represents the state of the object and $e(x)$ the state of the environment at time x. The simulation environment assures that all objects are executing these production rules in an "infinite" loop, *i.e.*, the execution cycles that simulate the passage of time.

In this execution cycle the production rules are executed in *parallel* and *simultaneously*. This means that during each execution cycle the data visible for each production rule do not change until all the production rules are executed.

Initially, the production rules are given explicitly for each object. These initial rules describe the basic interactions, and can be compared to the simulation of physical phenomena. Another set of production rules might be used to modify the object itself. These rules can be compared to a cell machine that interprets genetic information. If these production rules can be handled as data by other production rules, the system is capable of *self-modification*.

3.2 Objects

An object is parsed from its string representation (obj(ATTR$_1$, ATTR$_2$, ... , ATTR$_n$)), which describes the name of the object (obj), and its attributes in parentheses.

For example, an object might take the following

form.

```
obj(ATTR1=value, ATTR2=value,
    ..., ATTRn=value,
    RULE1=(obj :: cond → [ obj(...) ]),
    RULE2=(obj :: cond → [ obj(...) ]),
    ...
    '(RULE1(this), RULE2(this))
                        // Execution of rules
)
```

The above describes the initial state of an object with some attributes and two production rules. The execution of the production rules is defined by an explicit call to the rule with a parameter to define the object on which the rule will be applied. The parameter can be a reserved word (**this** meaning this object, or **up** meaning the upper level object) or the attribute name of a list of objects. (A detailed explanation will follow in section 3.4).

If we have a hierarchical representation of objects, we have the problem of defining whether the production rules are called before applying the production rules of sub-objects, or afterwards. This is solved in the current implementation by explicitly defining 'before' and 'after' execution of production rules. The execution order of multilevel production rules is shown in Figure 4.

In the following example (Figure 5) the execution order of the rules is as follows (|| means in parallel): RULE_1, (RULE_A || RULE_C), RULE_B, RULE_2.

3.3 Attributes

Each object has an attribute list, where each attribute has a name and a value (*e.g.*, a chemical compound with its concentration). A general description of an attribute is 'ATTR=value'. The attribute value can have a wide variety of predefined types with dynamic type resolution during execution. The predefined types include production rules, objects, arithmetic types (integer, double, *etc.*), geometrical types (point, segment, polygon, vector, *etc.*), and nested lists of the above. Attributes are modified by expression statements.

3.4 Interaction rules

Each interaction rule is described as a production rule. A production rule describes the conditions and the kind of action that will take place. Production rules are divided into the following types, according to the possible actions.

- Modification of internal state based on the internal state (cell machine)
- Modification of internal state based on the external state (cell membrane)
- Creation of a new object
- Deletion of an object

Below we explaine each type briefly.

3.4.1 Internal-state-based modification

These modifications, which have access only to the internal state of an object, have the following form of production rule.

```
RULE=(obj :: cond
          → obj(ATTR=expr(ATTR)) ),
...
'(RULE(this))
```

This means that the value of this object (indicated by the reserved word **this**) is given to the production rule as a value of *obj* and the attributes are modified according to the expression (ATTR=*expr*(ATTR)).

3.4.2 External-state-based modification

Access to the upper level values has the following form of production rule.

```
RULE=(obj, mobj :: cond
          → obj(ATTR=expr(mobj.ATTR)))
...
'(RULE(this, up))
```

The interpretation of the above rule is to call it with the value of this object (**this**) given to *obj*, and the value of the upper level object (indicated by the keyword **up**) given to *mobj*. In same cases (see examples in Figures 12 and tab:reproduction) we need to access not only the upper-level object, but also all the neighbors of the upper level object. Although this can be implemented by additional rules, we have a shortcut with the keyword **up*** meaning that the rule is called by all the upper-level given objects in turn, as the value to *mobj*.

Access to the sub-objects has the following form of production rule.

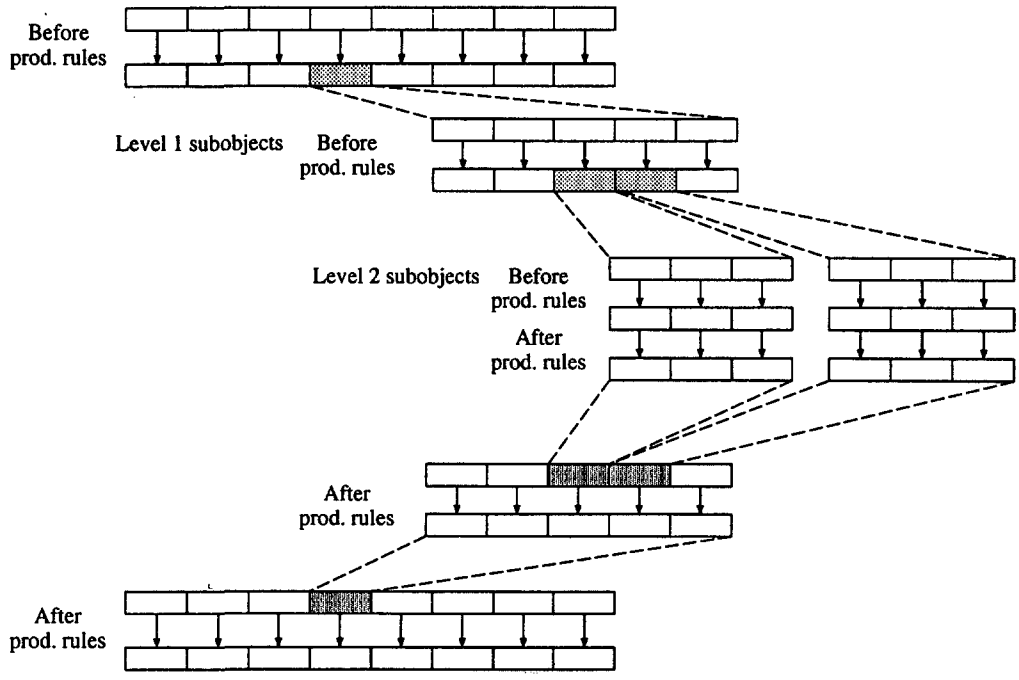


Figure 4: The execution of multilevel production rules. First, the 'before' rules are applied in parallel to the outmost object. This is repeated for each sub-object in parallel until there are no more sub-objects. Then the 'after' rules of the innermost sub-objects are applied and the sub-objects are rewritten into the upper level. This is continued until all 'after' rules are applied at all levels.

```
obj(RULE1=(obj :: cond → obj(...) ),
    RULE2=(obj :: cond → obj(...) ),
    SUBOBJS=[sobj1(RULEA=(obj :: cond → obj(...)),
                   RULEB=(obj :: cond → obj(...)),
                   ...
                   '(RULEA(this)),                    // Execution of before rules
                   '(RULEB(this))),                   // Execution of after rules
             sobj2(RULEC=(obj :: cond → obj(...)),
                   ...
                   '(RULEC(this))),                   // Execution of before rules
             ...],
    ...
    '(RULE1(this))                                    // Execution of before rules
    '(RULE2(this))                                    // Execution of after rules
)
```

Figure 5: An example of production rules to illustrate the execution order.

```
SUBOBJS=[ sobj(ATTR, ...), ...],
RULE=(obj, sobj :: cond
          → obj(ATTR=expr(sobj.ATTR)))
...
'(RULE(this, SUBOBJS))
```

The interpretation of the above rule is to call it with a value of this object (**this**) given to *obj*, and each sub-object (in SUBOBJS) as the value for *sobj*.

Both these mechanisms are illustrated by the example shown in Figure 6. The cells in the environment has access to the environment attribute ENVATTR through the production rule ADAPT. The environment object (*environment*) access of the attribute EFFECTATTR of each cell through the production rule EFFECT.

3.4.3 Creation of a new object

Creation of a new object takes place when there is an extra successor, as follows.

```
RULE=(obj :: cond
          → obj(ATTR=expr(ATTR)),
            obj_new(ATTR=expr(ATTR)) )
```

This rule has an extra successor, which is recognized by a different name (*i.e.*, $obj = obj$ results in only attribute modifications, but $obj \neq obj_new$ results in the creation of a new object). The default behavior is to copy the attribute values of the original object. However, the attributes can be modified within the same rule. Thus by modifying the attribute values we can achieve the following basic cell lineage types:

- The cell divides into two cells of same type $(A \rightarrow A, A)$
- The cell creates a new cell type $(A \rightarrow A, B)$
- The cell divides into two cells of different types $(A \rightarrow B, C)$

3.4.4 Deletion of an object

Deletion of an object is simple with the following rule.

```
RULE=(obj :: cond → ~obj() )
```

When we apply the rule we delete the objects where successor is preceded by '~' are deleted as a result of applying the rule. An alternative method would be to follow the principle of rewriting systems, *i.e.*, if the object is not rewritten by any rule, it will disappear. However, this would have required a default rule without any modification, and thus it was considered an unnecessary computational requirement.

3.5 Simulation of Physical Space

The basic idea that the environment cannot be distinguished from the system requires the modeling of physical phenomena to some extent. Currently, physical phenomena have been implemented by production rules similar to the rules controlling the self-modifications. This requires a complicated set of production rules to handle the physical properties of each object. Some of these properties, such as intersection in space, can be determined by built-in functions.

4 Morphogenesis of Organisms

The following example illustrates the MLIS language. With a simple morphogenesis process we can create a cellular structure. The cell divisions and differentiation are shown. In the final structure, some cells act as sensors, some cells as effectors, and some cells as neurons. The neuron cells start to grow connections that will terminate at other neurons, effectors and sensors. The created behavior will control the organisms in the environment and, based on this behavior, simple reproduction is demonstrated.

The examples given were executed by a serial implementation of the interpreter while the parallel implementation is under construction.

4.1 Divisions

Figure 7 describes how to create a simple morphogenesis. The description consists of *environment* object, which contains a list of organisms (ORGANISMS). Each organism (*organism*) is defined by a list of cells (CELLS). Initially, the list consists of description of a single cell (*cell*). In the example the cell has a production rule that defines when, and in what direction, to divide (DIVISION). Each of the divided cells will inherit the same production rule. Thus, the rule consists of five alternative branches, which are selected based on the

```
environment(ENVATTR,                                          // Factor effecting the cells
   CELLS = [ cell(ADAPTATTR,                                  // Factor effected by the environment
                  EFFECTATTR,                                 // Factor effecting the environment
                  ADAPT=(cell, env                            // Cell ⇐ Env
                           :: cond(cell.ADAPTATTR, env.ENVATTR)
                           → cell(ADAPTATTR=f(cell.ADAPTATTR, env.ENVATTR))
                         ),
                  '(ADAPT(this, up))
                ),                                            // End of cell description
             ...],
   EFFECT=(env, cell :: cond(env.ENVATTR, cell.EFFECTATTR)    // Env ⇐ Cells
                  → env(ENVATTR=f(ENVATTR, cell.EFFECTATTR)) ),
   '(EFFECT(this, CELLS))
)                                                             // End of environment description
```

Figure 6: An example of a production rule to illustrate the access of external attributes. The rule ADAPT accesses the attribute ENVATTR of the environment object (*environment*), which is assigned to the parameter *env* by using the keyword **up**. In contrast the EFFECT rule accesses each cell in CELLS (list of objects) that are assigned in turn to the parameter *cell*.

current stage of the cell (STAGE). The direction of possible divisions is given by a two-dimensional array of vectors (DIR).

Close study of the production rule reveals that the first branch divides the initial cell into 'up' and 'down' cells. The next branch will divide both of these cells into 'left' and 'right' cells. The four cells thus far created are capable of creating a structure that is symmetrical about the horizontal and vertical axes. The next three branches are used to create symmetrical quadrants, which differ only in the division direction. The first of the remaining rules gives a stem cell for the further cell lineage. The next branch starts a diagonal division lineage. The last rule creates a horizontal division lineage.

The cell divisions are shown in Figure 9. The development consists of eight cell divisions resulting in a 36-cell structure. The size of the structure is controlled by the number of diagonal and horizontal divisions (MAXDIV). This variable can be presented as an array similar to division direction (DIR), or as a function of the environment. This breaks the symmetry of the organisms as discussed below (4.3 *Environmental effect*).

4.2 Differentiation

During division the cells differentiate based on the production rule shown in Figure 8). The first branch of the production rule determines that the cell will end as an effector (muscle cell). The second branch will produce a sensor cell. The last branch will produce a neuron. The selection of the applied branch is based on the simple rule that "edge" cells will become effectors, "head" cells will become sensors, and "inter" cells will become neurons.

As shown in Figure 9 the cell size changes. This indicates that the cell becomes mature (STAGE= =mature). In the case of neural cells a slight rotation is used to distinguish them from immature cells (on the computer screen these are shown in color).

4.3 Environmental effect

The above examples demonstrate how the system is capable of morphogenesis of organisms based on simple genetic information. Let's assume that some of the control variables of the above production rules of cell divisions depend on the environment. For example, the value of MAXDIV can be defined based on some environmental factors, and produce the organism shown in Figure 10.

```
environment(...,                                          // Attributes of environment
   ORGANISMS=[                                            // Environment consists of organisms
      organism(...                                        // Attributes of organism
         CELLS=[...                                       // Organism consists of cells
            cell(POS=≪0.0,0.0≫, CNT, MAXDIV,
               DIR=[[≪0.0,10.0≫, ≪180.0,10.0≫], [≪90.0,10.0≫, ≪270.0,10.0≫],
                    [≪60.0,10.0≫, ≪120.0,10.0≫], [≪300.0,10.0≫, ≪240.0,10.0≫]],
                    RIGHT=1, LEFT=2, UP=3, DOWN=4,
               DIVIDE=(cell :: (cell.STAGE==stem)          // 'Up' and 'down' stem cells
                           → [ new(STAGE=stem1, POS+=DIR[2][1], LOC=UP),
                               cell(STAGE=stem1, LOC=DOWN) ]
                           :: (cell.STAGE==stem1)          // 'Left' and 'right' stem cells
                           → [ new(STAGE=stem2, POS+=DIR[1][1], SIDE=RIGHT),
                               cell(STAGE=stem2, SIDE=LEFT) ]
                           :: (cell.STAGE==stem2)          // Stem cell of quatro-body
                           → [ new(STAGE=stem3, POS+=DIR[1][SIDE]),
                               cell(STAGE=edge) ]
                           :: (cell.STAGE==stem3)          // Diagonal cell lineage
                           → [ new(STAGE=(CNT<MAXDIV?stem3:edge),
                                   POS+=DIR[LOC][SIDE]),
                               cell(STAGE=stem4) ]
                           :: (cell.STAGE==stem4)          // Horizontal cell lineage
                           → [ new(STAGE=(Cnt<MaxDiv?stem4:head),
                                   POS+=DIR[1][SIDE]),
                               cell(STAGE=inter) ]), /* DIVIDE */
               '(DIVIDE(this))
            ), ...], ...), ...], ...
) /* environment */
```

Figure 7: An example of production rules to generate cell divisions. ("≪ *angle*, *length* ≫" is a vector value; "... ? ... : ..." is a conditional statement "if ... then ... else ..."; "+=" is an addition to the previous value (here as a vector operation).)

5 Ontogenesis of Organisms

The transition from morphogenesis to ontogenesis is not clear. We refer here to the neural activities, namely the growth of the connections and the signal propagation between neurons, as ontogenetic phenomena. Thus far the signal propagation is not modeled to cause any permanent changes in the neural behavior, although this could easily be included in the model.

5.1 The growth of a neural network

When the cell structure has been formed, the neuron cells start to grow connections. The growth mechanism is similar to the cellular division mechanism. Each cell has a list of parts (CONNECTIONS), which contains the connections. Initially, the list contains a generic connection from which the other connections are initiated.

The generic connection has the production rules needed to model the growth. These rules are shown in Figure 11. The first rule (BUD) creates the initial connections with the necessary initial value of the genetic growth force (GF), *i.e.*, the direction in which the connection intends to grow.

The next rule (BRANCH) is a branching rule, which duplicates the connection by changing the direction of the growth force by, in this example, a constant factor (BRANCHANGLE). The condition for applying the branching rule is also determined by a constant factor (BRANCHCNT).

The growth of the connection is modeled by the next production rule (GROWTH. It contains three alternative execution branchs. The first branch calculates a new position (POS) by adding the ge-

```
cell(...
  DIFFER=(cell :: (cell.STAGE == edge) && (cell.CNT ≥ cell.MAXDIV)
                → [ cell(STAGE=mature, TYPE=effector, SHAPE *= 1.6) ]
                :: (cell.STAGE == head) && (cell.CNT ≥ cell.MAXDIV)
                → [ cell(STAGE=mature, TYPE=sensor, SHAPE *= 1.6) ]
                :: (cell.STAGE == inter) && (cell.CNT ≥ cell.MAXDIV)
                → [ cell(STAGE=mature, TYPE=neuron, R += 15.0) ]
  '(DIFFER(this))
) /* cell */
```

Figure 8: An example of a production rule that generates cell differentiations. (See explanation in text.)

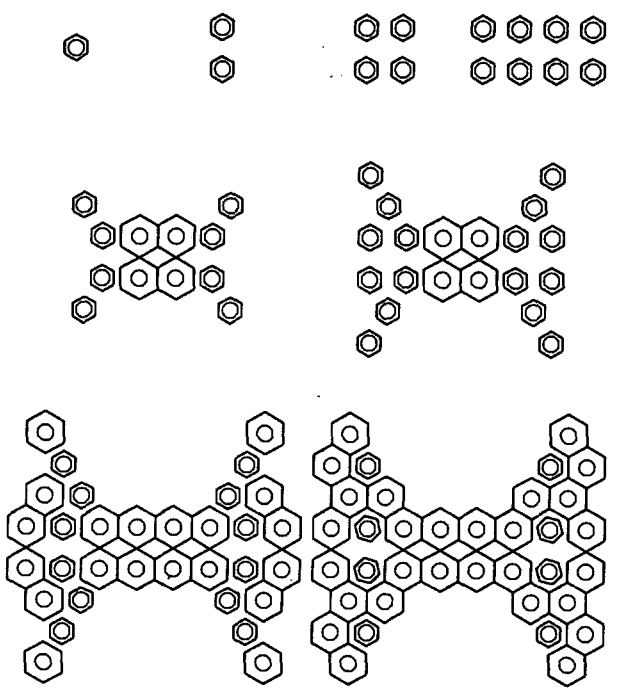


Figure 9: An example of the formation of a structure. Through multiple cell divisions and differentiations, the final form is gradually reached. A cell that no longer divides as is represented by a large cell or, in the case of a neural cell, by a slight rotation.

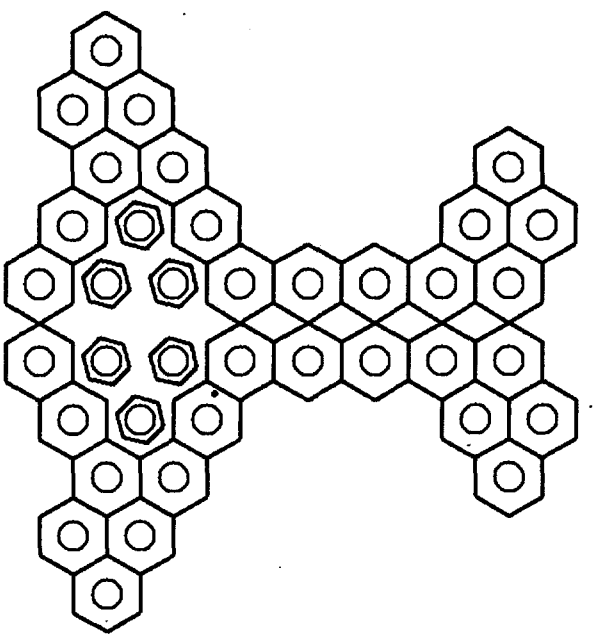


Figure 10: Variation of organisms by changing the value of a single control variable.

netic growth vector (GF). The new position is saved in the list (CONNPATH). The growth force is modeled to decrease as a function (f, a sigmoid function returning a value of less than one) of time (CNT).

The next branch is executed if the connection was not able to connect to any neuron by a specific number of growth steps (WITHDRAW). The withdrawal of the connection is implemented by repeated 'pop' operations until the list of connection growth steps (CONNPATH) is empty. When the list becomes empty the last branch will delete the connection.

5.1.1 Environmental effect

The above genetic growth of connections is insufficient to simulate realistic neural growth. We also need a targeting mechanism that includes physical and chemical factors. These can be implemented by the production rules shown in Figure 12.

The first rule (ATTRACTION) implements the chemical attraction of other cells. In order to make this work we need a target label (TARGET) for each that connection corresponds to the chemical diffusion labels (DIFF) of other cells. For each of the matching labels we calculate an attraction force (AF) as a function (f, as in Figure 13) of distance (| cell.Pos - Pos |) directed toward the attraction cell (*norm*(*cell*.POS - POS)).

In the above growth production rule (GROWTH) the new position calculation is modified to include the attraction force, and to suppress the genetic force in the vicinity of target cell.

5.1.2 Example of the growth of a neural network

In the following example we use a simplified model of the organisms. The cell positions are given explicitly as are the targeting labels, *i.e.*, what neuron will be connected to which sensors and effectors. The initial description is given in Figure 14. These values could be the result of the above described morphogenesis process, but due to the computational requirements, this has not yet been modeled.

An example of a created network is shown in Figure 15. The effecting forces include the simulation of mechanical collisions, and the chemical

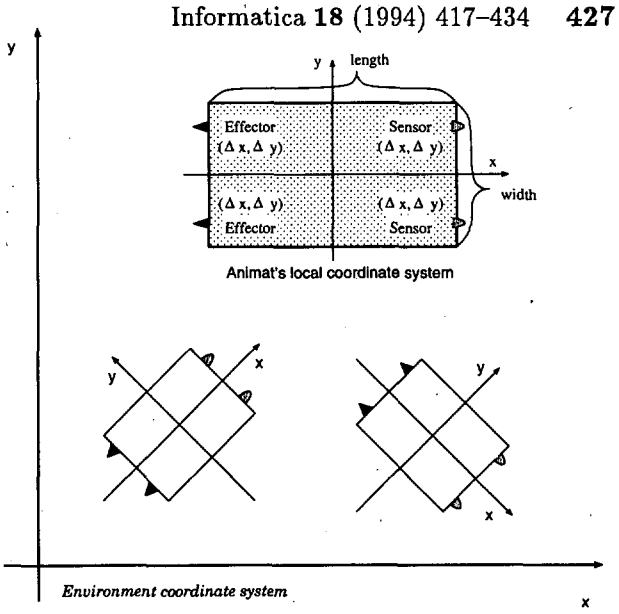


Figure 14: Each organism has its own local coordinate system. The length and width of an organism, and the positions of sensors and effectors are given relative to this.

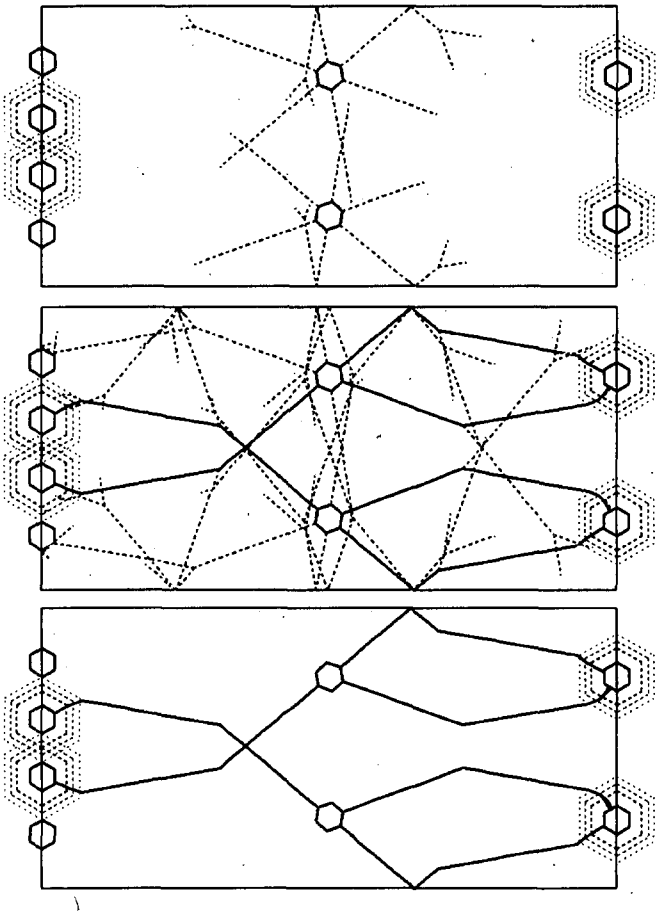


Figure 15: The neural cells grow connections that target other neurons, sensors and effectors. When to branch, and the branching angle are determined by the genetic rules. Three phases are shown: initial growth (top), initial withdrawal (middle), and after all unconnected connections are withdrawn (bottom).

```
cell(POS=≪ ... , ... ≫,                          // Cell as a neuron capable of growing connections
  CONNECTIONS=[
    generic(BRANCHCNT=10, BRANCHANGLE=[30.0, -30.0], WITHDRAW=0.1,
      DIR=[≪60.0, 5.0≫, ≪120.0, 5.0≫, ≪180.0, 5.0≫,
           ≪240.0, 5.0≫, ≪300.0, 5.0≫, ≪360.0, 5.0≫]
      DIV=1, CONNPATH=[], CONNECT=False,
      BUD=(conn, cell                                  // Start of a new connection
           :: (conn.DIV≤6)
           → [ new(GF=DIR[DIV], POS=cell.POS+GF, CONNPATH+=POS)
               conn(DIV++) ]),
      BRANCH=(conn                                     // A new connection by branching
           :: ((conn.CNT%conn.BRANCHCNT)==0)
           → [ new(GF=rotate(GF, BRANCHANGLE[1])),
               conn(GF=rotate(GF, BRANCHANGLE[2]) ]),
      GROWTH=(conn                                     // Growth of a connection
           :: (!conn.CONNECT) && (|conn.GF|>conn.WITHDRAW)
           → [ conn(POS+=GF, CONNPATH+=POS, GF*=f(CNT) ]
           :: (!conn.CONNECT) && (conn.CONNPATH≠[])
           → [ conn(POS=first(CONNPATH), CONNPATH=rest(CONNPATH) ]
           :: (!conn.CONNECT) && (conn.CONNPATH==[])
           → [ ~conn() ]
      '(BUD(this, up), BRANCH(this, up), GROWTH(this))
  ) /* cell */
```

Figure 11: An example of production rules that produce genetic growth of connections. The first rule (BUD starts new connections with initial values. The second rule BRANCH creates new connections through a branching operation. The third rule GROWTH grows, withdraws, or removes the connections according to the WITHDRAW factor.

```
ATTRACTION=(conn, cell                                // Chemical gradient field
          :: (conn.TARGET==cell.DIFF)
          → [ conn(AF+=f(|cell.POS - POS|)*norm(cell.POS-POS)) ]),
GROWTH=(conn ...POS += f(AF)*GF + AF ...),
COLLISION=(conn, obj                                  // Mechanical collision
        :: (!conn.CONNECT) && (conn.TARGET≠obj.DIFF) &&
           (inter(obj.SHAPE, conn.CONNPATH))
        → [ conn(GF=rotate(GF, -2.0*interAng()),
                 POS=interP()+norm(GF)*|interP()-first(CONNPATH)|,
                 CONNPATH=push(POS, push(interP(), rest(CONNPATH)))) ]
                                                      // ... or hitting the target
        :: (!conn.CONNECT) && (conn.TARGET==obj.DIFF) &&
           (inter(obj.SHAPE, conn.CONNPATH)) && (obj.TYPE == eff)
        → [ conn(CONNECT=connect(conn, obj), TYPE=axon) ])
        :: (!conn.CONNECT) && (conn.TARGET==obj.DIFF) &&
           (inter(obj.SHAPE, conn.CONNPATH)) && (obj.TYPE == sen)
        → [ conn(CONNECT=connect(obj, conn), TYPE=dend) ])
...,
'(ATTRACTION(this, up*), COLLISION(this, up*))
```

Figure 12: Examples to show production rules to model the mechanical (COLLISION) and the chemical (ATTRACTION) effect on growth. Notice that the last production rule also creates a 'synaptical connection' if the grown connection hits the target. This is implemented by making an internal link between the connected objects.

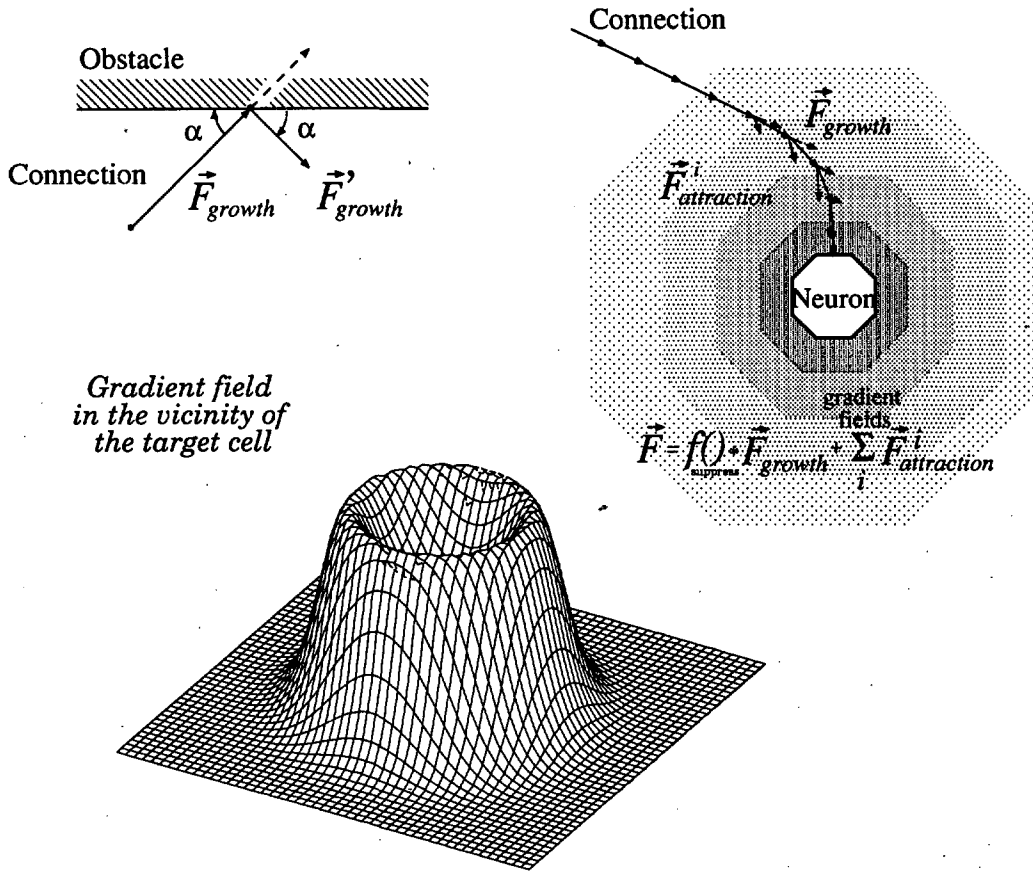


Figure 13: The attraction field created by the chemical diffusion of a cell located at the middle of picture. (The depression of the field close to cell is used to model the slowdown of growth when approaching the target cell. A more realistic approach might be to use two gradient fields: one for the rough approaching, and one for the final approaching.)

gradient field on which the connection is "climbing".

5.2 Signal propagation in the network

After the network is created the same modeling method is used to propagate signals with in it. Signal propagation is governed by the production rules shown in Figure 16.

Signal propagation is divided into three phases: from cell or axon to dendrite, from dendrite to neuron (the generic connection), from neuron to axon. These are rules at the sub-object level of the cell, and implemented by three separate rules: PROPAGATE1, PROPAGATE2, and PROPAGATE3. The separation is needed to assure the correct order of signal propagation.

After signal propagation at the sub-object level of the cell we still need to get the signal to the cellular level objects, namely to the effectors. This is implemented with a separate rule (PROPAGATE).

The example shows signal propagation with no learning included. It would be easy to include a threshold, connection weights, or other internal modifications in the above rules.

5.2.1 Example of created behavior

In order to observe the behavior of the above neural network, it should be placed in an environment where sensors generate signals into the network, and the created effector activity moves the organism according to the received signals. To simulate this, the above production rules can be extended to model the sensor activity as well as the movement of organisms according to the generated forces. This is discussed in [11] and here only a short summary is given in Figure 17.

6 Phylogenesis of Organisms

The organisms behaving in the environment could own a behavior that leads to reproduction. As a result of reproduction, a new initial description is created. In this process some genetic variations can take place. The initial cell with its genetic information is placed in the environment and closes the evolutionary loop. The evolution of organisms is thought to be a result of this process.

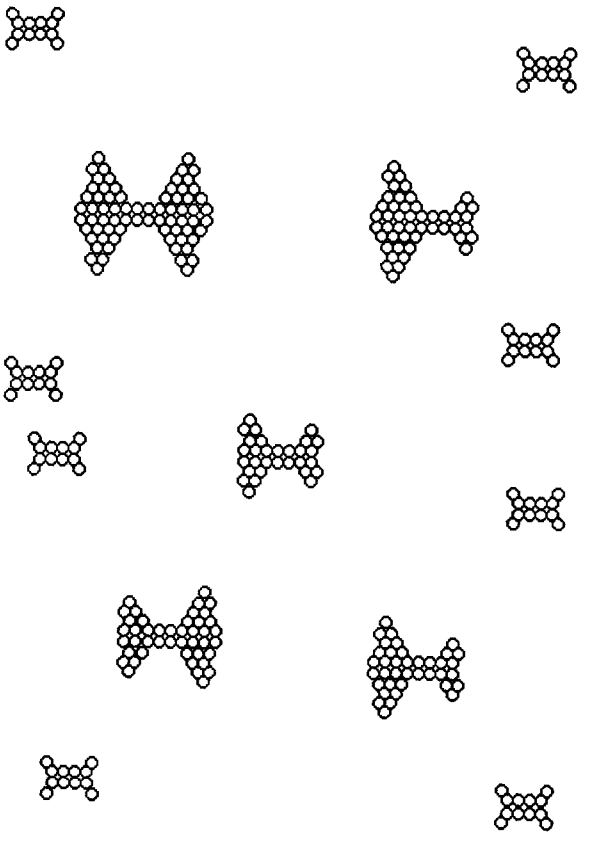


Figure 19: Reproduction: An organism (center) ejects four seeds into the environment, each of which will give rise to a new organism with slightly different control variables. These new organisms in turn each create two new organisms.

6.1 Reproduction

The first example describes reproduction that is biologically plausible given the structure used in section 4 *Morphogenesis of Organisms*. In Figure 18, the production rule (EJECT 'ejects' an initial cell into the environment resulting in the growth of a new organism. By identifying some control variables as genetic information with mutations, we can create a great variation of shapes in the environment. This is seen in Figure 19, where MAXDIV is used as mutable genetic information.

The above example reveals the problem of creating a selection of favorable structures based on local interactions. With no behavior included in the model, there is little change of creating a selection mechanism. In order to use behavior in the selection process, we use the example introduced in section 5 *Ontogenesis of Organisms*.

Here we define the genetic code as the positions

```
cell(...,                                                    // Cell capable of propagating signals
  PARTS=[generic(TYPE=neuro, ...,
       PROPAGATE1=(obj1, obj2                                  // From cell or axon to dendrite
                  :: (obj1.TYPE == dend) → [ obj1( SIGNAL = obj2.SIGNAL ) ])
       PROPAGATE2=(obj1, obj2                                  // From dendrite to neuron
                  :: (obj1.TYPE == neuro) → [ obj1( SIGNAL += obj2.SIGNAL ) ]),
       PROPAGATE3=(obj1, obj2                                  // From neuron to axon
                  :: (obj1.TYPE == axon) → [ obj1( SIGNAL = obj2.SIGNAL ) ]),
       ...,
       '(..., PROPAGATE1(this, Connected()),
            PROPAGATE2(this, Connected()),
            PROPAGATE3(this, Connected()), ...),
  PROPAGATE=(obj1, obj2                                        // From axon to effector
            :: (obj1.TYPE == eff) → [ obj1(SIGNAL += obj2.SIGNAL) ])
  '(..., PROPAGATE(this, Connected()), ...) )] )
```

Figure 16: Example to show signal propagation in the network. The rules PROPAGATE1, PROPAGATE2, and PROPAGATE3 are used to propagate the signal from sensor cell to dendrite, from dendrite to neuron, and from neuron to axon, respectively. The rule PROPAGATE is used to transfer a signal to the effectors. The function *Connected()* is used to give a list of objects that are connected to the current object.

of the explicitly given sensors and effectors. Production rules to select organisms for reproduction and to introduce genetic variations are shown in Figure 20. The selection is based on the intersection of the physical shapes. When an intersection is detected, a new offspring inherits its parents genetic information is created.

6.2 Evolution

The current implementation does not allow simulation of evolution with a complicated behavior model. However, a simple example is used to demonstrate how the behavior can enforce itself. Although the example does not include any creation of new behavior the previously described non-linear development process can provide it.

In the following example we allow a single organism to move in the environment, and another to follow it. Whenever the following organism reaches the target, a new organism is created. In this example, only the location of the sensors and effectors is varied. Gradually an organism capable of better following behavior will evolve. This is shown in Figure 21.

7 Conclusion

In this paper we have discussed *computational evolution* rather than evolutionary computation. By computational evolution we mean a model of evolution at a detailed level with a goal to better understand the evolutionary process itself. The evolutionary process is greatly affected by the environment in a way that is not yet fully understood.

The usual technique for modeling evolutionary systems ignores the developmental phase. It is important to encode and transcribe the genetic information in a way that can provide a non-linear developmental process. As has been observed [13], biological life has not developed through small steps only, but has also developed through larger jumps. This cannot easily be explained by random mutation alone.

By modeling the non-linear developmental process, a single variation in the genetic information can create a completely different phenotype. If this variation is caused by environmental factors, the probability that this will happen at the same time in several individuals is great. Thus, the system would be capable of generating new species for different environmental niches.

In the last part of this paper we presented a de-

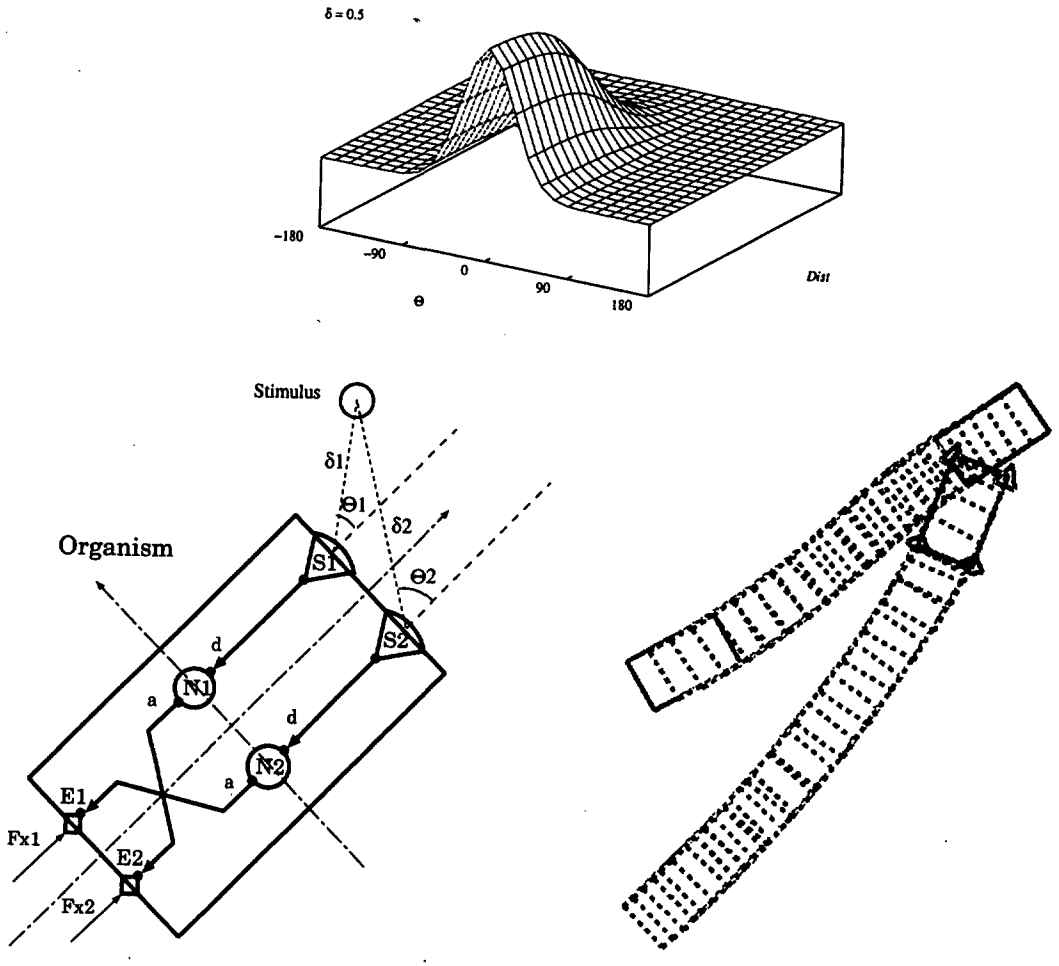


Figure 17: Sensor activity is modeled based on a simple model. The signal generated by the sensor is a function of distance and angle (at top) to the source of stimulus. The simplified chart (left) shows how the signal is propagated to effectors, which generate a force vector proportional to the signal. This results in tracking behavior (right).

```
EJECT=(cell :: (cell.TYPE == eject) &&                          // Rule to eject a seed
          (cell.CNT ≥ cell.REPRODAGE) && (cell.CNT%cell.EJECTRATE == 0)
        → [ seed(INITED=False, MAXDIV = random(5, 6, 7),
              POS += random(10.0, 25.0) *
                  cell.DIR[random(UP, DOWN)][random(LEFT, RIGHT)]),
            cell() ]
    ) // EJECT
```

Figure 18: An example of production rules to implement reproduction. The production rule EJECT creates a seed cell in the environment. This cell is capable of growing new structure similar to the original one.

```
organism(                                          // Organism capable of creating offsprings
    SENPOS1=(...), SENPOS2=(...),
    EFFPOS1=(...), EFFPOS2=(...),
    MATE=(org1, org2
        :: (inter(org1.SHAPE, org2.SHAPE))
        → [ offspring(SENPOS1=mutate(crossover(org1.SENPOS1, org2.SENPOS1)),
                      SENPOS2=mutate(crossover(org1.SENPOS2, org2.SENRPOS)),
                      EFFPOS1=mutate(crossover(org1.EFFPOS1, org2.EFFPOS1)),
                      EFFPOS2=mutate(crossover(org1.EFFPOS2, org2.EFFPOS2)),
                      POS=translate(POS, OFFSETVECTOR), ...) ]),
    '(..., MATE(this, up*), ...)
)
```

Figure 20: An example of production rules to implement reproduction. The production rule (*Mate*) tests all other organisms (*up**) against the calling organism (*this*) to determine whether there is an intersection in their physical shapes. If they intersect, a new offspring is created. The condition can include tests for sex, maturity, *etc.* There should also be some threshold variable to insure that only one offspring is created at a time. The crossover function (*crossover*) can be an average function, and the mutation function (*mutate*) can be a random change of the value, for example, in the range of $[-0.5, 0.5]$ units.

scription language that provides a computational basis for experiments on non-linear developmental and evolutionary processes. The language is based on production rules that simulate the interactions between objects. This means that our focus is not on the structure itself, but on the interactions between objects that define a higher level object. This is conceptually similar to *autopoiesis* theory [4].

The examples given at the end of the paper are presented more to demonstrate the simulation language and to illustrate the power of the non-linear construction process. The actual analysis of the creation of new species and, in general, the evolvability of the modeled organisms, is left for future work.

The main merit of the presented modeling method is its capability of covering all the needed detailed phases needed for the simulation of evolution. This can also be cited as the main critics. Too much is included in one model and the modeling of populations is not possible with current computer technology. However, the method allows us to focus on the essential: *local interaction*. Thus, it can be used as a powerful tool for modeling self-organization in order to achieve deeper understanding of the factors that cause evolution.

References

[1] Jean Claude Brinquier. *Conversations with Jean Piaget.* The University of Chicago Press, 1980.

[2] Stuart A. Kauffman. *The Origins of Order – Self-organization and selection in evolution.* Oxford University Press, New York, Oxford, 1993.

[3] John R. Koza. *Genetic Programming: On the programming of computers by means of natural selection.* The MIT Press, 1992.

[4] H. R. Maturana and F. J. Varela. *Autopoiesis and Cognition: The Realization of the Living.* Reidel, 1980.

[5] Setsuo Ohsuga and Jari Vaario. A study of artificial life as a model of automatic model building. In Hannu Jaakkola, Hannu Kangassalo, Tadahiro Kitahashi, and András Márkus, editors, *Information Modelling and Knowledge Bases V*, pages 1–22. IOS Press, 1994.

[6] Jean Piaget. *Le comportment, moteur de l'évolution.* Colloection Idées. Gallinard, Paris, 1976.

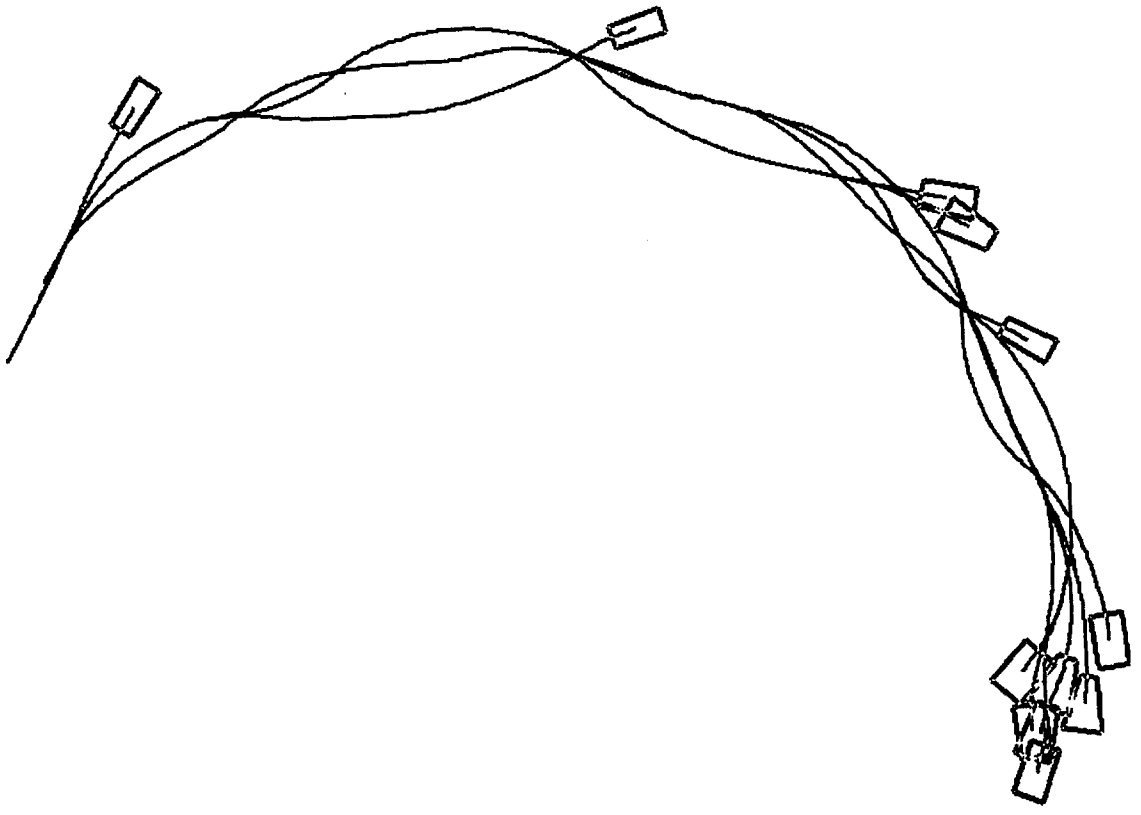


Figure 21: An example of evolution where initially a single organism follows target and, when reaching it generates an offspring with some mutations that also begins to follow the same target. Gradually a structure capable of better following behavior will evolve. The simulation does not include any explicitly given fitness function.

[7] Premyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants.* Springer-Verlag, 1990.

[8] Olliver C. Rieppel. *Fundamentals of Comporative Biology.* Birkhäuser-Verlag, Basel, 1988.

[9] Jari Vaario. *An Emergent Modeling Method for Artificial Neural Networks.* PhD thesis, The University of Tokyo, 1993.

[10] Jari Vaario. Artificial life as constructivist AI. *Journal of SICE (Society of Instrument and Control Engineers)*, 33(1):65—71, 1994.

[11] Jari Vaario, Koichi Hori, and Setsuo Ohsuga. Toward evolutionary design of autonomous systems. *The International Journal in Computer Simulation*, 1994. (to appear).

[12] Jari Vaario and Setsuo Ohsuga. On growing intelligence. In Georg Dorffner, editor, *Neural Networks and a New AI.* Chapman & Hall, London, 1994.

[13] Robert Wesson. *Beyond Natural Selection.* The MIT Press, 1993.

An Experimental Study of N-Person Iterated Prisoner's Dilemma Games

Xin Yao and Paul J. Darwen
Department of Computer Science
University College, The University of New South Wales
Australian Defence Force Academy
Canberra, ACT, Australia 2600

Keywords: genetic algorithms, prisoner's dilemma, learning, generalisation

Edited by: Matjaž Gams
Received: November 9, 1994 **Revised:** November 28, 1994 **Accepted:** December 6, 1994

The Iterated Prisoner's Dilemma game has been used extensively in the study of the evolution of cooperative behaviours in social and biological systems. There have been a lot of experimental studies on evolving strategies for 2-player Iterated Prisoner's Dilemma games (2IPD). However, there are many real world problems, especially many social and economic ones, which cannot be modelled by the 2IPD. The n-player Iterated Prisoner's Dilemma (NIPD) is a more realistic and general game which can model those problems. This paper presents two sets of experiments on evolving strategies for the NIPD. The first set of experiments examine the impact of the number of players in the NIPD on the evolution of cooperation in the group. Our experiments show that cooperation is less likely to emerge in a large group than in a small group. The second set of experiments study the generalisation ability of evolved strategies from the point of view of machine learning. Our experiments reveal the effect of changing the evolutionary environment of evolution on the generalisation ability of evolved strategies.

1 Introduction

The 2-player Iterated Prisoner's Dilemma game (2IPD) is a 2×2 non-zerosum noncooperative game, where "non-zerosum" indicates that the benefits obtained by a player are not necessarily the same as the penalties received by another player and "noncooperative" indicates that no preplay communication is permitted between the players [1, 2]. It has been widely studied in such diverse fields as economics, mathematical game theory, political science, and artificial intelligence.

In the Prisoner's Dilemma, each player has a choice of two operations: either cooperate with the other player, or defect. Payoff to both players is calculated according to Figure 1. In the Iterated Prisoner's Dilemma (IPD), this step is repeated many times, and each player can remember previous steps.

While the 2IPD has been studied extensively for more than three decades, there are many real

	Cooperate	Defect
Cooperate	R, R	T, S
Defect	S, T	P, P

Figure 1: The payoff matrix for the 2-player prisoner's dilemma game. The values S, P, R, T must satisfy $T > R > P > S$ and $R > (S + T)/2$. In 2-player Iterated Prisoner's Dilemma (2IPD), the above interaction is repeated many times, and both players can remember previous outcomes.

world problems, especially many social and economic ones, which cannot be modelled by the 2IPD. Hardin [3] described some examples of such problems. More examples can be found in Colman's book [1](pp.156–159). The n-player Iterated Prisoner's Dilemma (NIPD) is a more realistic and general game which can model those problems. In comparing the NIPD with the 2IPD, Davis *et al.* [4](pp.520) commented that

> The N-player case (NPD) has greater generality and applicability to real-life situations. In addition to the problems of energy conservation, ecology, and overpopulation, many other real-life problems can be represented by the NPD paradigm.

Colman [1](pp.142) and Glance and Huberman [5, 6] have also indicated that the NIPD is "qualitatively different" from the 2IPD and that "... certain strategies that work well for individuals in the Prisoner's Dilemma fail in large groups."

The n-player Prisoner's Dilemma game can be defined by the following three properties [1](pp.159):

1. each player faces two choices between cooperation (C) and defection (D);
2. the D option is dominant for each player, i.e., each is better off choosing D than C no matter how many of the other players choose C;
3. the dominant D strategies intersect in a deficient equilibrium. In particular, the outcome if all players choose their non-dominant C strategies is preferable from every player's point of view to the one in which everyone chooses D, but no one is motivated to deviate unilaterally from D.

Figure 2 shows the payoff matrix of the n-player game.

A large number of values satisfy the requirements of Figure 2. We choose values so that, if n_c is the number of cooperators in the n-player game, then the payoff for cooperation is $2n_c - 2$ and the payoff for defection is $2n_c + 1$. Figure 3 shows an example of the n-player game.

With this choice, simple algebra reveals that if N_c cooperative moves are made out of N moves of an n-player game, then the average per-round payoff a is given by:

$$a = 1 + \frac{N_c}{N}(2n - 3) \quad (1)$$

This lets us measure how common cooperation was just by looking at the average per-round payoff.

There has been a lot of research on the evolution of cooperation in the 2IPD using genetic algorithms and evolutionary programming in recent years [7, 8, 9, 10, 11, 12]. Axelrod [7] used genetic algorithms to evolve a population of strategies where each strategy plays the 2IPD with every other strategy in the population. In other words, the performance or fitness of a strategy is evaluated by playing the 2IPD with every other strategy in the population. The environment in which a strategy evolves consists of all the remaining strategies in the population. Since strategies in the population are constantly changing as a result of evolution, a strategy will be evaluated by a different environment in every generation. All the strategies in the population are co-evolving in their dynamic environments. Axelrod found that such dynamic environments produced strategies that performed very well against their population. Fogel [11] described similar experiments, but used finite state machines to represent strategies and evolutionary programming to evolve them.

However, very few experimental studies have been carried out on the NIPD in spite of its importance and its qualitative difference from the 2IPD. This paper presents two sets of experiments carried out on the NIPD. We first describe our experiment setup in Section 2. Then we investigate the impact of the number of players in the Prisoner's Dilemma game on the evolution of cooperation in Section 3. We are mainly interested in two questions here: (1) whether cooperation can still emerge from a larger group, and (2) whether it is more difficult to evolve cooperation in a larger group. The evolution of strategies for the NIPD can be regarded as a form of machine learning using the evolutionary approach. An important issue in machine learning is generalisation. Section 4 of this paper discusses the generalisation issue associated with co-evolutionary learning and presents some experiments with different evolutionary environments. Finally, Section 5 concludes with some remarks and future research directions.

Number of cooperators among the remaining $n-1$ players

player A	0	1	2	...	$n-1$
C	C_0	C_1	C_2	...	C_{n-1}
D	D_0	D_1	D_2	...	D_{n-1}

Figure 2: The payoff matrix of the n-player Prisoner's Dilemma game, where the following conditions must be satisfied: (1) $D_i > C_i$ for $0 \leq i \leq n-1$; (2) $D_{i+1} > D_i$ and $C_{i+1} > C_i$ for $0 \leq i < n-1$; (3) $C_i > (D_i + C_{i-1})/2$ for $0 < i \leq n-1$. The payoff matrix is symmetric for each player.

Number of cooperators among the remaining $n-1$ players

player A	0	1	2	...	$n-1$
C	0	2	4	...	$2(n-1)$
D	1	3	5	...	$2(n-1)+1$

Figure 3: An example of the N-player game.

2 Experiment Setup

2.1 Genotypical Representation of Strategies

We use genetic algorithms to evolve strategies for the NIPD. The most important issue here is the representation of strategies. We will use two different representations, both of which are look-up tables that give an action for every possible contingency.

One way of representing strategies for the NIPD is to generalise the representation scheme used by Axelrod [7]. In this scheme, each genotype is a lookup table that covers every possible history of the last few steps. A history in such a game is represented as a binary string of ln bits, where the first l bits represent the player's own previous l actions (most recent to the left, oldest to the right), and the other $n-1$ groups of l bits represent the previous actions of the other players. For example, during a game of 3IPD with a remembered history of 2 steps, $n = 3, l = 2$, one player might see this history:

$n = 3, l = 2$: Example history `11 00 01`

The first l bits, `11`, means this player has defected (a "1") for both of the previous $l = 2$ steps. The previous steps of the other players are then listed in order: the `00` means the first of the other players cooperated (a "0") on the previous l steps, and the last of the other players cooperated (0) on the most recent step, and defected (1) on the step before, as represented by `01`.

For the NIPD remembering l previous steps, there are 2^{ln} possible histories. The lookup table genotype therefore contains an action (cooperate "0" or defect "1") for each of these possible histories. So we need at least 2^{ln} bits to represent a strategy. At the beginning of each game, there are no previous l steps of play from which to look up the next action, so each genotype should also contain its own extra bits that define the presumed pre-game moves. The total genotype length

is therefore $2^{ln}+ln$ bits. We will use this genotype for the first set of results below, Figure 5 through to Figure 8.

This Axelrod-style representation scheme, however, suffers from two disadvantages. First, it does not scale well as the number of players increases. Second, it provides more information than is necessary by telling which of the other players cooperated or defected, when the only information needed is how many of the other players cooperated or defected. Such redundant information had reduced the efficiency of the evolution greatly in our experiments with this representation scheme. To improve on this, we use a new representation scheme which is more compact and efficient.

In our new representation scheme, each individual is regarded as a set of rules stored in a look-up table that covers every possible history. As a game that runs for, say, 500 rounds would have an enormous number of possible histories, and as only the most recent steps will have significance for the next move, we only consider every possible history over the most recent l steps, where l is less than 4 steps. This means an individual can only remember the l most recent rounds. Such a history of l rounds is represented by:

1. l bits for the player's own previous l moves, where a "1" indicates defection, a "0" cooperation; and

2. another $l \log_2 n$ bits for the number of cooperators among the other $n-1$ players, where n is the number of the players in the game. This requires that n is a power of 2.

For example, if we are looking at 8 players who can remember the 3 most recent rounds, then one of the players would see the history as:

History for 8 players, 3 steps: `001 111 110 101`
(12 bits)

Here, the `001` indicates the player's own actions: the most recent action (on the left) was a "0", indicating cooperation, and the action 3 steps ago (on the right), was a "1", i.e., defection. The `111` gives the number of cooperators among the other 7 players in the most recent round, i.e., there were $111_2 = 7$ cooperators. The `101` gives the number of cooperators among the other 7 players 3 steps ago, i.e., there were $101_2 = 5$ cooperators. The most recent events are always on the left, previous events on the right.

In the above example, there are $2^{12} = 2048$ possible histories. So 2048 bits are needed to represent all possible strategies. In the general case of an n-player game with history length l, each history needs $l + l \log_2 n$ bits to represent and there are $2^{l+l \log_2 n}$ such histories. A strategy is represented by a binary string that gives an action for each of those possible histories. In the above example, the history `001 111 110 101` would cause the strategy to do whatever is listed in bit 1013, the decimal number for the binary 001111110101.

Since there are no previous l rounds at the beginning of a game, we have to specify them with another $l(1 + \log_2 n)$ bits. Hence each strategy is finally represented by a binary string of length $2^{l+l \log_2 n} + l(1 + \log_2 n)$.

2.2 Genetic Algorithm Parameters

For all the experiments presented in this paper, the population size is 100, the mutation rate is 0.001, and the crossover rate is 0.6. Rank-based selection was used, with the worst performer assigned an average of 0.75 offspring, the best 1.25 offspring.

2.3 A Typical Run

A tyical run with four players with a history 1 ($n = 4, l = 1$) is shown in Figure 4. At each generation, 1000 games of the 4-player Iterated Prisoner's Dilemma are played, with each group of 4 players selected randomly with replacement. Each of these 1000 games lasts for 100 rounds. Starting from a random population, defection is usually the better strategy, and the average payoff plummets initially. As time passes, some cooperation becomes more profitable. We will examine more results in detail later.

3 Group Size of the NIPD

This section discusses the impact of group size, i.e., the number of players in the NIPD, on the evolution of cooperation and presents some experimental results. It is well-known that cooperation can be evolved from a population of random strategies for the 2IPD. Can cooperation still be

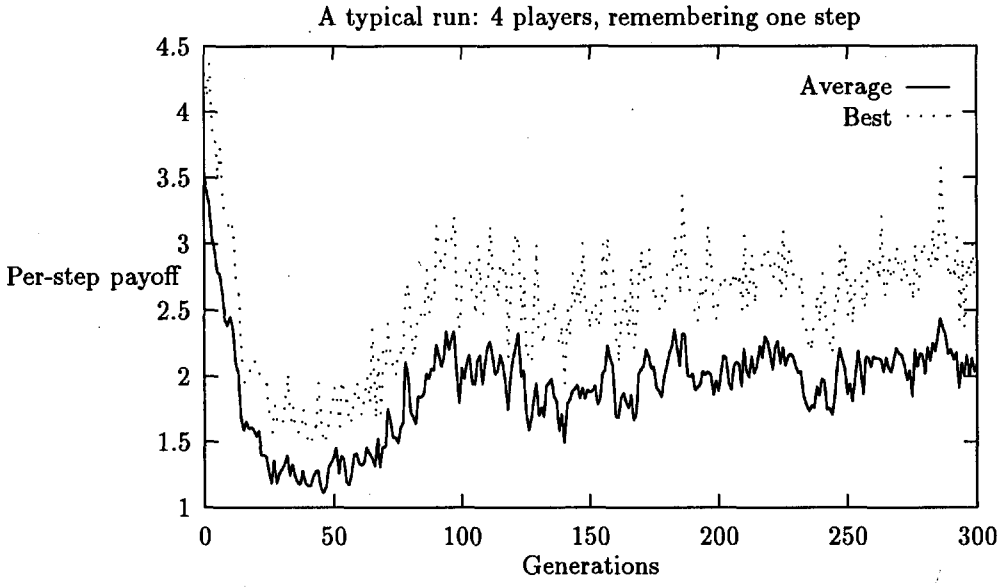


Figure 4: This shows the average and best payoff at each generation for a population of 100 individuals. Each individual is a strategy.

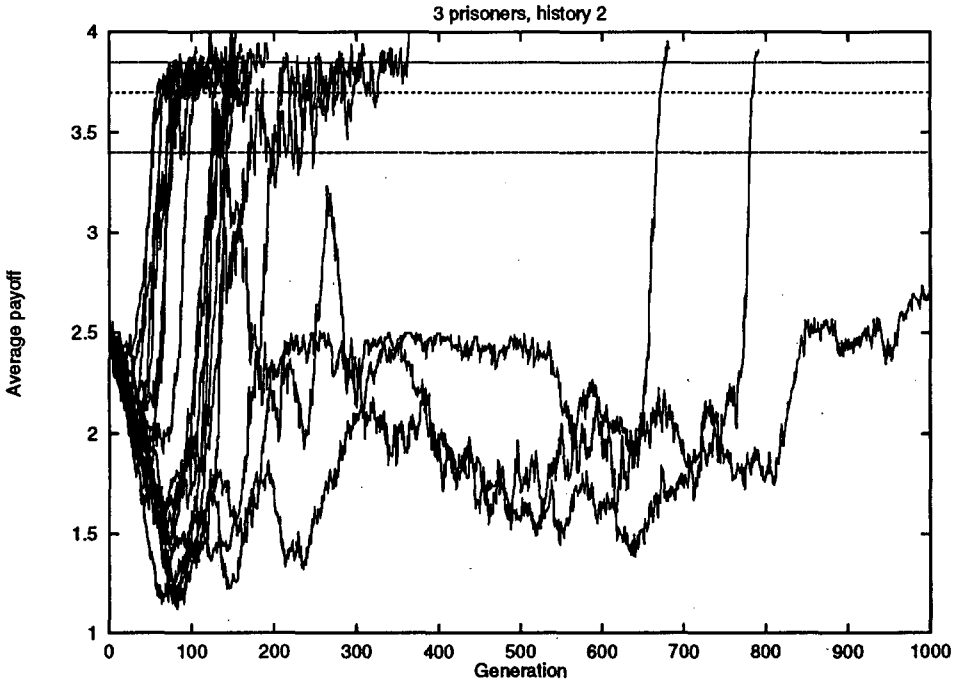


Figure 5: For the 3-player prisoner's dilemma with a history of 2, cooperation almost always emerges. Only 1 out of 20 runs fail to reach 95% cooperation using Axelrod's representation shceme.

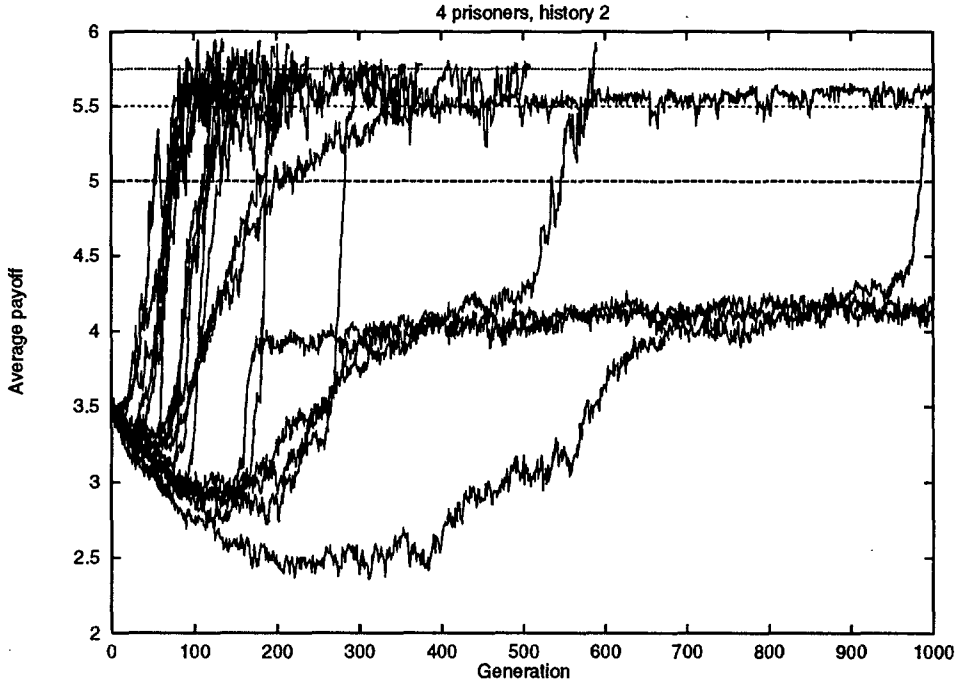


Figure 6: For the 4-player prisoner's dilemma with a history of 2, cooperation almost always emerges. Only 4 out of 20 runs fail to reach 95% cooperation using Axelrod's representation shceme.

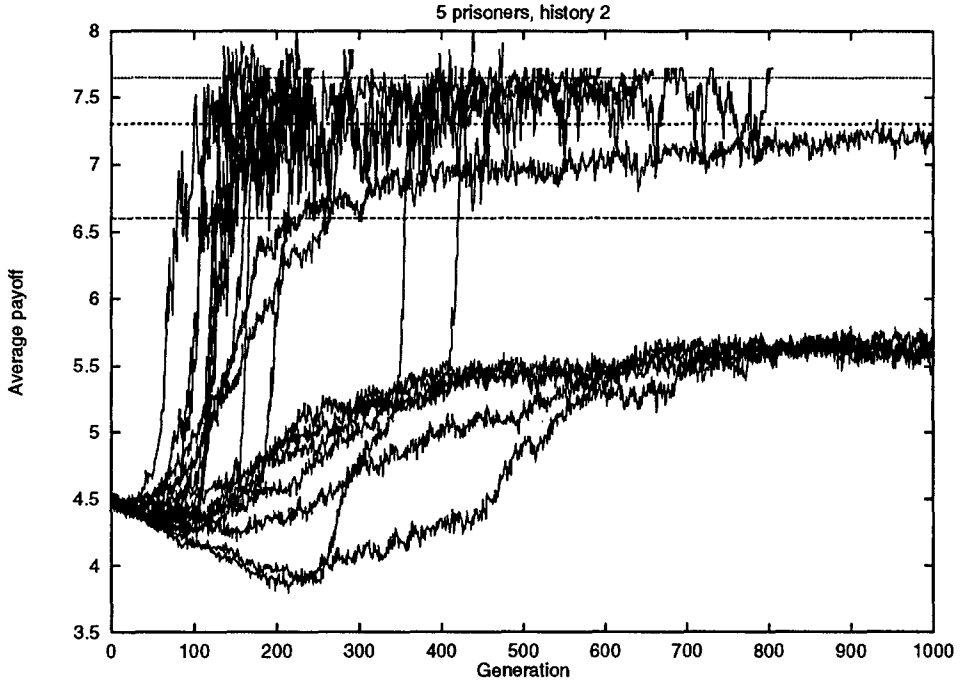


Figure 7: For the 5-player prisoner's dilemma with a history of 2, cooperation almost always emerges. 6 out of 20 runs fail to reach 80% cooperation using Axelrod's representation shceme.

evolved from a population of strategies for the NIPD where the number of players is greater than 2? If the answer is yes, does the group size affect the evolution of cooperation in the NIPD?

Using the Axelrod-style genotype described above, we carried out a series of experiments with the 3IPD, 4IPD, 5IPD, and 6IPD games. In each of the following runs, the program stopped when more than 5 generations passed with the average payoff above the 95% cooperation level. Figure 5 shows the results of 20 runs of the 3IPD game with history length 2: out of 20 runs, there is only 1 which fails to reach 95% cooperation. Figure 6 shows the results of 20 runs of the 4IPD game with history length 2: 4 out of 20 runs fail to reach the 95% cooperation level, but only 1 of those fails to reach 80% cooperation. Figure 7 shows the results of 20 runs of the 5IPD game with history length 2: 6 out of 20 runs do not reach the 80% cooperation level. Figure 8 shows the results of 20 runs of the 6IPD game with history length 2: 9 out of 20 runs stay below the 80% cooperation level.

Figures 5 through 8 demonstrate that the evolution of cooperation becomes less likely as group size increases. Nonetheless, cooperation still emerges most of the time. As Axelrod's representation scheme used in those figures does not scale well with the group size, we use the second representation scheme described in Section 2 to carry out experiments with larger groups.

We have carried out a series of experiments with the 2IPD, 4IPD, 8IPD, and 16IPD games. Figure 9 shows the results of 10 runs of the 2IPD game with history length 3. Out of 10 runs, there are only 3 which fail to reach 90% cooperation and only 1 which goes to almost all defection. Figure 10 shows the results of 10 runs of the 4IPD game with history length 3, where some of the runs reach cooperation but more than half of the 10 runs fail to evolve cooperation. Figure 11 shows the results of 10 runs of the 8IPD game with history length 2, where none of the runs reach cooperation. Figure 12 shows the population bias in the runs in Figure 11, to demonstrate that those populations have pretty much converged. Figure 13 shows 10 runs of the 16IPD game.

These results confirm that cooperation can still be evolved in larger groups, but it is more difficult to evolve cooperation as the group size increases. Glance and Huberman [5, 6] have arrived at a similar conclusion using a model based on many particle systems. We first suspected that the failure to evolve cooperation in larger groups was caused by larger search spaces and insufficient running time since more players were involved in 8IPD and 16IPD games. This is, however, not the case. The search space of the 8IPD game with history length 2 is actually smaller than that of the 4IPD game with history length 3. To confirm that the failure to evolve cooperation is not caused by insufficient running time, we examined the convergence of the 8IPD game. Figure 12 shows that at generation 200 the population has mostly converged for all the 10 runs.

It is worth mentioning that the evolution of cooperation using simulations does depend on some implementation details, such as the genotypical representation of strategies and the values used in the payoff matrix. So cooperation may be evolved in the 8IPD game if a different representation scheme and different payoff values are used. Although we cannot prove it vigorously, we think for any representation scheme and payoff values there would always be an upper limit on the group size over which cooperation cannot be evolved. Our experimental finding is rather similar to some phenomena in our human society, e.g., cooperation is usually easier to emerge in a small group of people than in a larger one.

4 Co-Evolutionary Learning and Generalisation

The idea of having a computer algorithm learn from its own experience and thus create expertise without being exposed to a human teacher has been around for a long time. For genetic algorithms, both Hillis [13] and Axelrod [7] have attempted co-evolution, where a GA population is evaluated by how well it performs against itself or another GA population, starting from a random population. Expertise is thus bootstrapped from nothing, without an expert teacher. This is certainly an promising idea, but does it work? So far, no-one has investigated if the results of co-evolutionary learning are robust, that is, whether they generalise well? If a strategy is produced by a co-evolving population, will that strategy perform well against opponents never seen by that

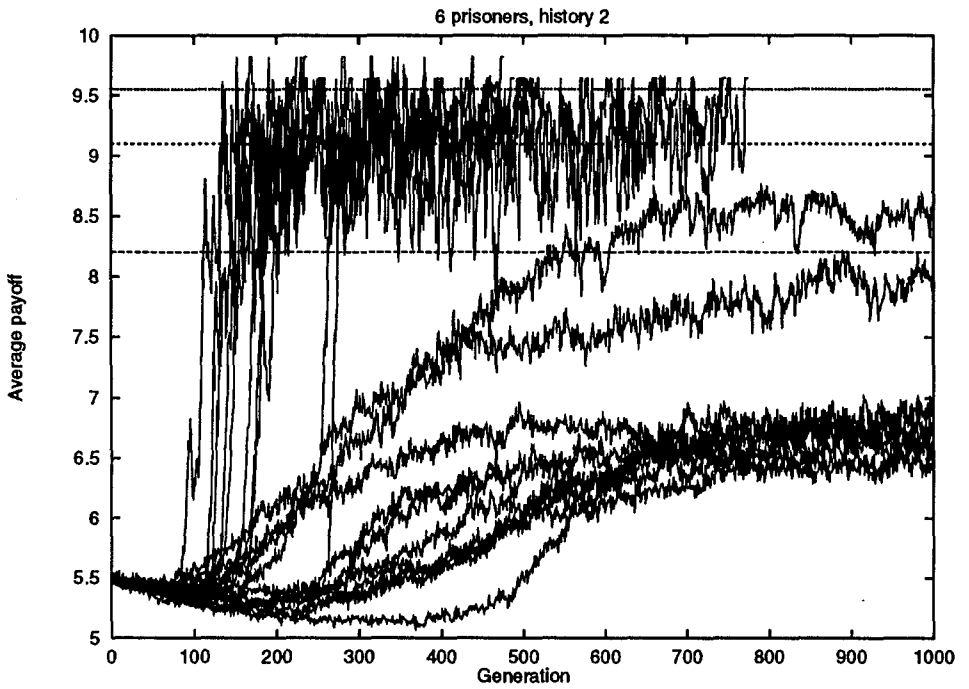


Figure 8: For the 6-player prisoner's dilemma with a history of 2, cooperation almost always emerges. 9 1 out of 20 runs fail to reach 80% cooperation using Axelrod's representation shceme.

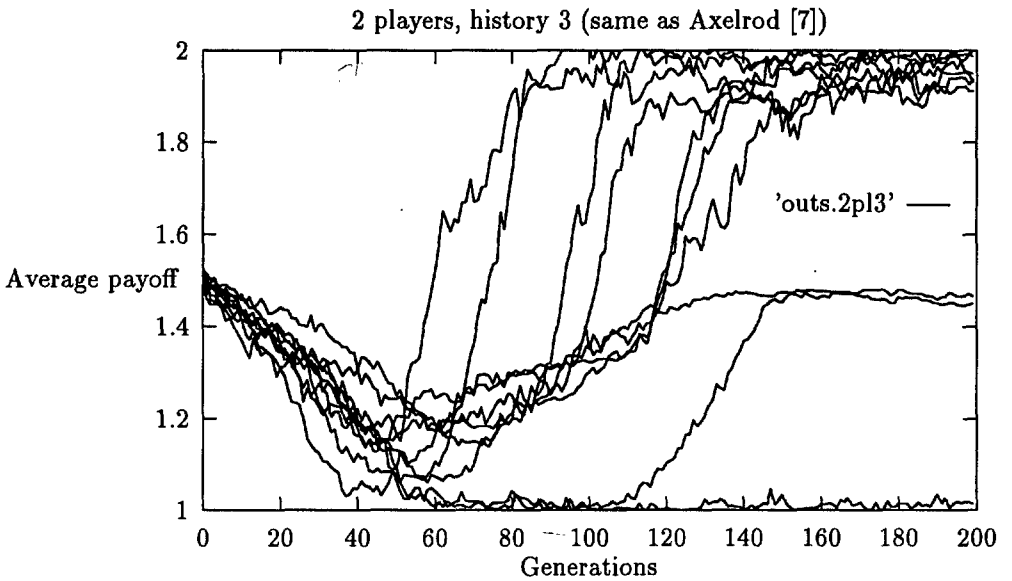


Figure 9: For 2-player prisoner's dilemma with a history of 3, cooperation emerges most of the time. Only 3 out of 10 runs fail to reach 90% cooperation, and only 1 run goes to almost all defection.

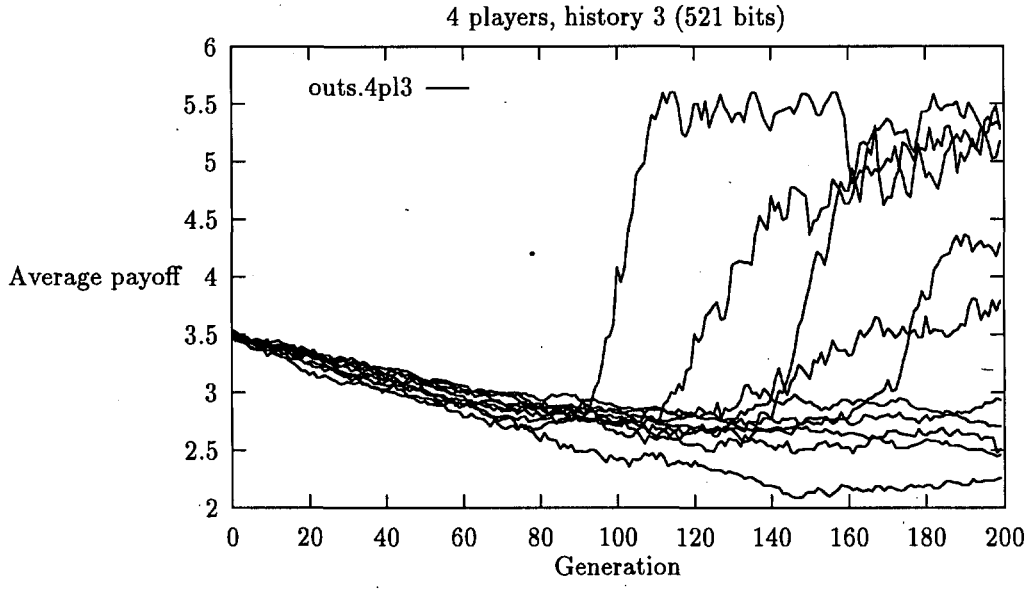


Figure 10: For 10 runs of 4-player prisoner's dilemma with a history of 3, cooperation breaks out some of the time.

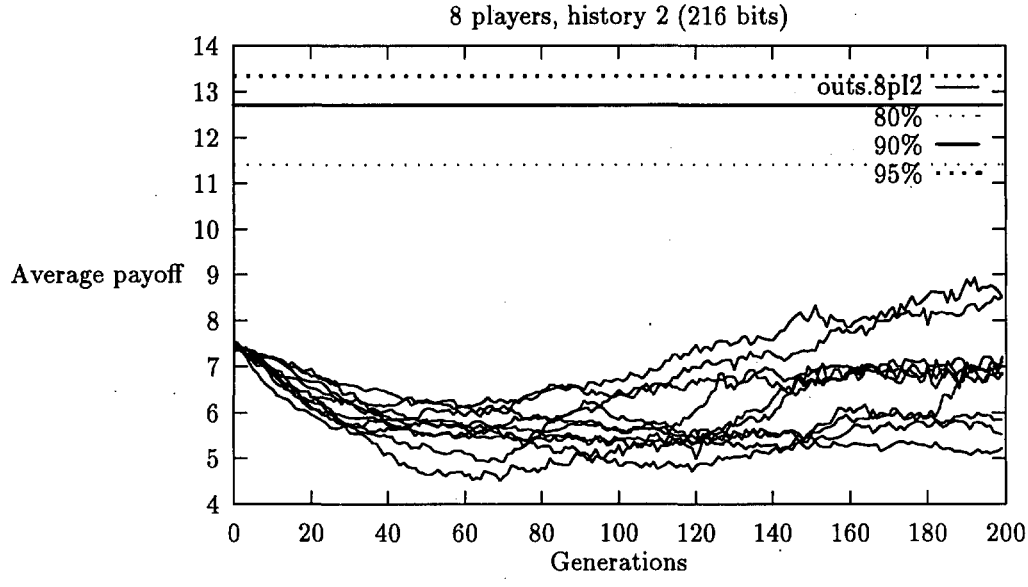


Figure 11: For 10 runs of 8-player prisoner's dilemma with a history of 2, cooperation never emerges. The horizontal lines at the top show the 95%, 90%, and 80% levels of cooperation. To demonstrate that these runs have converged, figure 12 shows the bias of the populations.

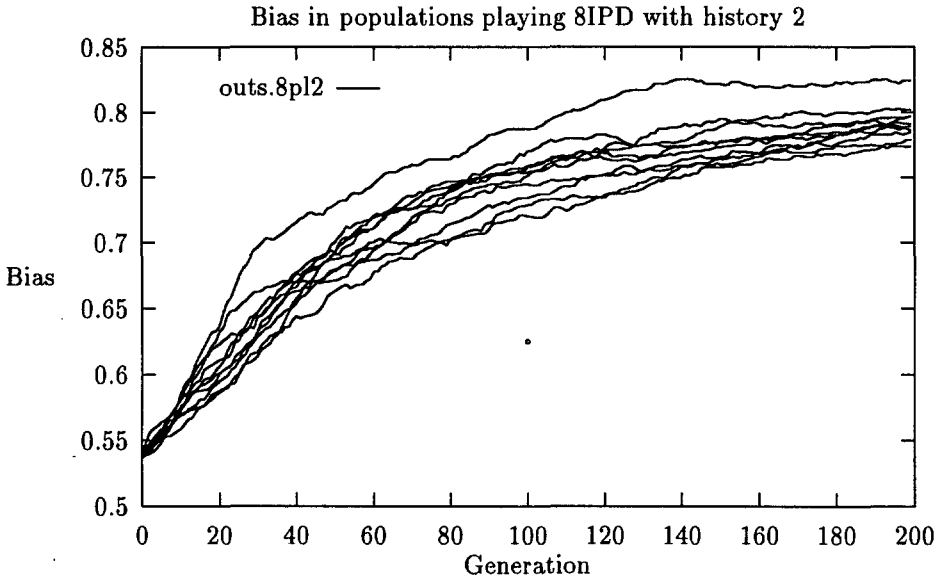


Figure 12: In 10 runs of 8-player prisoner's dilemma with a history of 2, where cooperation never emerges, the bias demonstrates that the populations have converged. Bias is the average proportion of the most prominent value in each position. A bias of 0.75 means that, on average, each bit position has converged to either 75% "0" or 75% "1".

population? In order to investigate this issue, we need to pick the best strategies produced by the co-evolutionary learning system and let them play against a set of test strategies which had not been seen by the co-evolutionary system. This section describes some experiments which test the generalisation ability of co-evolved strategies for the 8IPD game with history length 1.

4.1 Test Strategies

The unseen test strategies used in our study should be of reasonable standard and representative, that is, they are neither very poor (or else they will be exploited by their evolved opponents) nor very good (or else the will exploit their evolved opponent). We need unseen strategies that are adequate against a large range of opponents, but not *the* best.

To obtain such strategies, we did a limited enumerative search to find the strategies that performed best against a large number of random opponents. As most random opponents are very stupid, beating many random opponents provides a mediocre standard of play against a wide range of opponents. We limited this search to manageable proportions by fixing certain bits in a strategy's genotype that seemed to be sensible, such as always defecting after every other strategy defects. The top few strategies found from such a limited enumerative search are listed in Table 1.

4.2 Learning and Testing

We have compared three different methods for implementing the co-evolutionary learning system. The three methods differ in the way each individual is evaluated, i.e., which opponents are chosen to evaluate an individual's fitness. The three methods are

1. Choosing from among the individuals in the GA population, i.e., normal co-evolution of a single population like Axelrod's implementation [7];

2. Choosing from a pool made of the evolving GA population and the best 25 strategies from the enumerative search, which remain fixed;

3. Choosing from a pool made of the evolving GA population and the best 25 strategies from the enumerative search, but the proba-

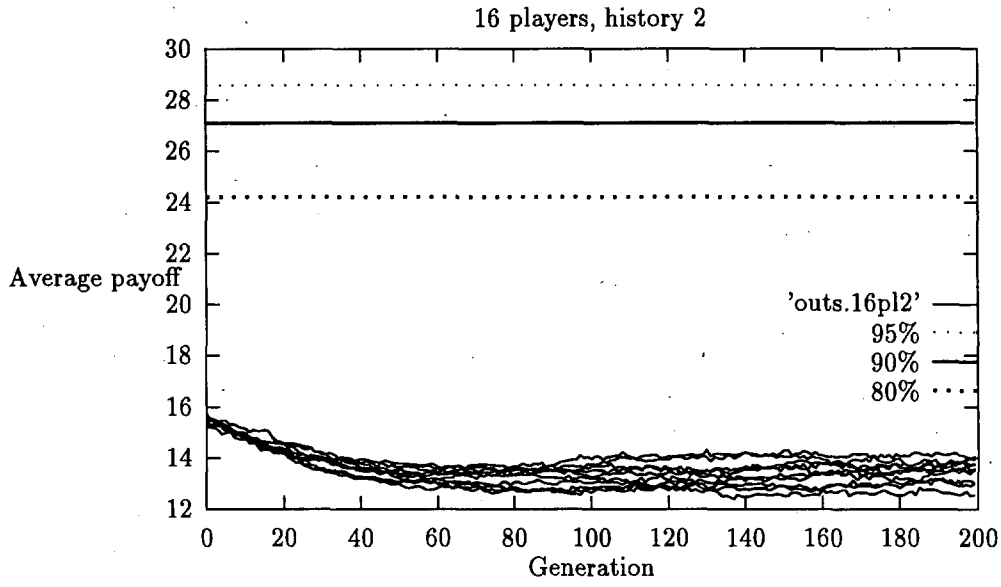


Figure 13: For 10 runs of 16-player prisoner's dilemma with a history of 2, cooperation never emerges. The horizontal lines at the top show the 95%, 90%, and 80% levels of cooperation.

Mean	Std Dev	Decimal	Binary genotype
8.100	0.083	1026040	1111 1010 0111 1111 1000
8.093	0.083	1022965	1111 1001 1011 1111 0101
8.091	0.083	1018871	1111 1000 1011 1111 0111
8.088	0.083	1032181	1111 1011 1111 1111 0101
8.088	0.083	1020921	1111 1001 0011 1111 1001
8.082	0.083	1028087	1111 1010 1111 1111 0111
8.077	0.083	1023990	1111 1001 1111 1111 0110
8.076	0.083	1037305	1111 1101 0011 1111 1001
8.076	0.083	1017846	1111 1000 0111 1111 0110

Table 1: Top few strategies from a partial enumerative search for strategies that play well against a large number of random opponents. This provides unseen test opponents to test the generalisation of strategies produced by co-evolution. The first 4 bits were fixed to "1", as were the eleventh through sixteenth bits. Virtually all of the best 50 strategies started by cooperating.

bility of choosing one of the 25 is four times higher.

For each of these, we obtained the best 25 strategies from the last generation of the GA, and tested it against a pool made up of both the seen and unseen enumerative search strategies, 50 in all.

4.3 Experimental Results

For each of the three evaluation methods, Tables 2 through 4 show the performance of the best strategies from the GA's last generation against opponents from (1) themselves, and (2) a pool made up of both the seen and unseen strategies from the enumerative search.

4.4 Discussion

Table 2 demonstrates that the co-evolution with the 8IPD produces strategies that are not very cooperative, as also demonstrated in Figure 11 earlier. Since the 8IPD is a game where it is easy to get exploited, co-evolution will first create strategies that can deal with non-cooperative strategies. The evolved strategies in Table 2 are cautious with each other and are not exploited by the unseen strategies from the enumerative search.

Adding fixed but not very cooperative strategies to the GA's evaluation procedure has a surprising effect. The evolved strategies in Tables 3 and 4 can cooperate well with other cooperators without being exploited by the strategies from the enumerative search, half of which it has never seen before. That is, normal co-evolution produces strategies which don't cooperate well with each other, and are not exploited by unseen non-cooperative strategies. Co-evolution with the addition of extra non-cooperative strategies gives more general strategies that do cooperate well with each other, but are still not exploited by unseen non-cooperative strategies. The experimental results also seem to indicate that the evolved strategies learn to cooperate with other cooperators better while maintaining their ability in dealing with non-cooperative strategies when the evolutionary environment contains a higher proportion of extra fixed strategies.

5 Conclusion

This paper describes two sets of experiments on the NIPD. The first set of experiments on the group size of the NIPD demonstrate that cooperation can still be evolved in the n-player IPD game where $n > 2$. However, it is more difficult to evolve cooperation as the group size increases. There are two research issues here which are worth pursuing; one is the upper limit of the group size over which cooperation cannot be evolved, the other is the quantitative relation between the group size and the time used to evolve cooperation. Glance and Huberman [5, 6] have addressed these two issues, but did not give a complete answer.

The second set of experiments in this paper deals with an important issue in co-evolutionary learning — the generalisation issue. Although the issue is the main theme in machine learning, very few people in the evolutionary computation community seem to be interested in it or address the issue explicitly and directly. We have presented some experimental results which show the importance of the environments in which each individual is evaluated, and their effects on generalisation ability.

References

[1] A. M. Colman, *Game Theory and Experimental Games*, Pergamon Press, Oxford, England, 1982.

[2] A. Rapoport, Optimal policies for the prisoner's dilemma, Technical Report 50, The Psychometric Lab., Univ. of North Carolina, Chapel Hill, NC, USA, July 1966.

[3] G. Hardin, The tragedy of the commons, *Science*, 162:1243–1248, 1968.

[4] J. H. Davis, P. R. Laughlin, and S. S. Komorita, The social psychology of small groups, *Annual Review of Psychology*, 27:501–542, 1976.

[5] N. S. Glance and B. A. Huberman, The outbreak of cooperation, *Journal of Mathematical Sociology*, 17(4):281–302, 1993.

Normal co-evolution, no extra strategies in evaluation.
GA strategies play against themselves.

		Mean	Stdv	Stdv of mean	Mean of opponents
8pl1 (35% cooperative) against itself					
0	11100001111011010011	7.240	3.978	0.126	6.296
1	11000001111011011011	7.285	3.985	0.126	6.392
2	11100000111111110011	7.335	3.052	0.097	8.434
3	01100010111111010011	7.258	3.202	0.101	7.889
4	11100101111111110011	7.180	4.160	0.132	5.883
5	01100000111111010011	7.000	3.090	0.098	8.111
6	11100101111111010011	7.171	4.125	0.130	6.127
7	01100001111111010011	7.241	4.027	0.127	6.286
8	00100000111111010110	7.165	3.122	0.099	8.341
9	01100100111111110110	7.706	3.523	0.111	7.949
10	10100000111111110001	7.274	3.083	0.097	8.395

GA strategies play against unseen strategies from enumerative search.

		Mean	Stdv	Stdv of mean	Mean of opponents
0	11100001111011010011	5.525	2.330	0.074	5.340
1	11000001111011011011	5.627	2.421	0.077	5.502
2	11100000111111110011	5.605	2.568	0.081	5.027
3	01100010111111010011	5.087	2.064	0.065	5.419
4	11100101111111110011	5.283	2.210	0.070	4.532
5	01100000111111010011	5.477	2.547	0.081	6.337
6	11100101111111010011	5.116	1.877	0.059	4.473
7	01100001111111010011	5.392	2.370	0.075	5.378
8	00100000111111010110	5.385	2.531	0.080	6.530
9	01100100111111110110	5.146	2.271	0.072	5.237
10	10100000111111110001	5.461	2.383	0.075	4.900

Table 2: Results of ordinary co-evolution, with no extra strategies during the GA evaluation. The GA strategies manage some cooperation among themselves, and hold their own against strategies they have not seen before.

Co-evolution, with addition of 25 fixed strategies from enumerative search.
GA strategies play against themselves.

		Mean	Stdv	Stdv of mean	Mean of opponents
0	11111000011111110100	11.678	1.715	0.054	11.965
1	11111000011111110100	11.706	1.553	0.049	11.994
2	11111000011111110110	11.440	1.603	0.051	11.922
3	11111000011111111110	11.721	1.581	0.050	12.027
4	11111000111111110100	13.264	2.521	0.080	10.636
5	11111000011111111110	11.714	1.584	0.050	12.025
6	11111000011111110110	11.420	1.669	0.053	11.895
7	11111000011111110100	11.678	1.705	0.054	11.985
8	11011000001111110100	11.618	1.781	0.056	11.958
9	11111000011111111111	11.670	1.688	0.053	11.974
10	11111000011111110100	11.649	1.697	0.054	11.973

GA strategies play against pool of 25 seen and 25 unseen strategies from enumerative search.

		Mean	Stdev	Stddev of mean	Mean of opponents
0	11111000011111110100	5.209	3.212	0.102	5.634
1	11111000011111110100	5.494	3.451	0.109	5.828
2	11111000011111110110	5.152	2.771	0.088	5.934
3	11111000011111111110	5.600	3.561	0.113	5.907
4	11111000111111110100	5.619	2.929	0.093	4.629
5	11111000011111111110	5.336	3.369	0.107	5.724
6	11111000011111110110	4.971	2.541	0.080	5.741
7	11111000011111110100	5.447	3.481	0.110	5.791
8	11011000001111110100	5.591	3.276	0.104	5.923
9	11111000011111111111	5.245	3.200	0.101	5.673
10	11111000011111110100	5.392	3.341	0.106	5.771

Table 3: Adding 25 fixed strategies to the evaluation procedure, along with the 100 co-evolving GA individuals, causes the GA to produce strategies that can cooperate more with each other, but are not exploited by the more non-cooperative strategies from the enumerative search.

Co-evolution, with the addition of 25 fixed strategies, which are 4 times as likely to be selected into the group of 8 players for 8IPD.

GA strategies play against themselves.

		Mean	Stdev	Stddev of mean	Mean of opponents
0	11111000011111110010	12.575	1.737	0.055	12.740
1	11111000011111110011	12.468	1.939	0.061	12.641
2	10111000011111010010	12.400	2.130	0.067	12.593
3	11111000011111111110	12.557	1.864	0.059	12.709
4	11111000011111110111	12.556	1.488	0.047	12.820
5	11111000011111010110	12.490	1.454	0.046	12.772
6	10111000011111110011	12.392	2.087	0.066	12.568
7	11111001011111111111	13.204	2.457	0.078	10.713
8	11111000011111111111	12.551	1.852	0.059	12.700
9	11111000011111110010	12.560	1.904	0.060	12.718
10	11111000011111110010	12.494	1.835	0.058	12.669

Best 25 strategies from GA search play against a pool of (1) 25 best from enumerative search, and (2) 25 unseen strategies from enumerative search. Note there is little diversity in the GA population.
GA strategies play against pool of 25 seen and 25 unseen strategies from enumerative search.

		Mean	Stdev	Stddev of mean	Mean of opponents
0	11111000011111110010	5.209	3.212	0.102	5.634
1	11111000011111110011	5.494	3.451	0.109	5.828
2	10111000011111010010	5.635	3.120	0.099	6.217
3	11111000011111111110	5.600	3.561	0.113	5.907
4	11111000011111110111	5.187	2.835	0.090	5.966
5	11111000011111010110	5.132	2.762	0.087	5.910
6	10111000011111110011	5.375	3.159	0.100	5.753
7	11111001011111111111	5.447	3.481	0.110	5.788
8	11111000011111111111	5.422	3.340	0.106	5.765
9	11111000011111110010	5.245	3.200	0.101	5.673
10	11111000011111110010	5.392	3.341	0.106	5.771

Table 4: Increasing the importance of the extra 25 fixed strategies causes the co-evolutionary GA to produce strategies that are even more cooperative among themselves, but are still not exploited by the unseen strategies of the enumerative search.

[6] N. S. Glance and B. A. Huberman, The dynamics of social dilemmas, *Scientific American*, pages 58–63, March 1994.

[7] R. Axelrod, The evolution of strategies in the iterated prisoner's dilemma, In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, chapter 3, pages 32–41. Morgan Kaufmann, San Mateo, CA, 1987.

[8] D. M. Chess, Simulating the evolution of behaviors: the iterated prisoners' dilemma problem, *Complex Systems*, 2:663–670, 1988.

[9] K. Lindgren, Evolutionary phenomena in simple dynamics, In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II: SFI Studies in the Sciences of Complexity, Vol. X*, pages 295–312, Reading, MA, 1991. Addison-Wesley.

[10] D. B. Fogel, The evolution of intelligent decision making in gaming, *Cybernetics and Systems: An International Journal*, 22:223–236, 1991.

[11] D. B. Fogel, Evolving behaviors in the iterated prisoner's dilemma, *Evolutionary Computation*, 1(1):77–97, 1993.

[12] P. J. Darwen and X. Yao, On evolving robust strategies for iterated prisoner's dilemma, In X. Yao, editor, *Proc. of the AI'93 Workshop on Evolutionary Computation*, pages 49–63, Canberra, Australia, November 1993. University College, UNSW, Australian Defence Force Academy.

[13] W. Daniel Hillis, Co-evolving parasites improve simulated evolution as an optimization procedure, In *Santa Fe Institute Studies in the Sciences of Complexity, Volume 10*, pages 313–323. Addison-Wesley, 1991.

Benchmarking Indicates Relevance of Multiple Knowledge

Matjaž Gams
Jožef Stefan Institute, Jamova 39, 61000 Ljubljana, Slovenia
Phone: +386 61 125 91 99, Fax: +386 61 219 677
E-mail: matjaz.gams@ijs.si

Keywords: artificial intelligence, multiple knowledge, multistrategy learning

Edited by: Anton P. Železnikar
Received: March 17, 1994 **Revised:** November 11, 1994 **Accepted:** November 15, 1994

Over the last 7 years, detailed measurements of available learning systems were performed on two real-life medical domains with the purpose to verify the importance of multiple knowledge. The performance of the combined system GINESYS, consisting of an artificial intelligence and a statistical method, was analysed with and without multiple knowledge and by varying the number of learning examples, the amount of artificially added noise, the impurity and the error estimate functions. These measurements and those of other researchers indicate that multiple knowledge can provide essential improvements. Measurements also indicate that improvements over "one-level" or monostrategy knowledge representation representations are quite common in real-life noisy and incomplete domains.

1 Introduction

In easing the bottleneck of knowledge acquisition in expert systems (Harmon et al. 1988), automatic knowledge construction from examples has proven useful in many practical tasks. Quite often, examples are described in terms of attributes and their values and each example belongs to a certain class. The task of the system is to induce concept descriptions from examples. First systems were designed for exact domains like chess end-games and constructed trees (ID3 - Quinlan 1983) or lists of rules (AQ11 - Michalski & Larson 1983). But in many real-life domains (Gams & Karalič 1989), because of noise or incomplete description (Manago & Kodratoff 1987) specialised mechanisms have to be applied. In noisy domains, longer rules (or longer branches in trees) perform better on learning examples while truncated rules (pruned tees with shorter branches) perform better on unseen examples. On the basis of this principle, the second group of inductive systems emerged (CART - Breiman et al. 1984; AQ15 - Michalski et al. 1986; ASSISTANT - Kononenko 1985; CN2 - Clark & Niblett 1989; C4 - Quinlan 1987). Around five years ago the third group of systems began emerging (GINESYS - Gams 1988; 1989; LOGART - Cestnik, Bratko 1988; new CN2 - Clark & Boswell 1991), based on the explicit use of multiple knowledge.[1] Each of these groups of systems usually achieves better performance than previous. Better performance of multiple knowledge systems was especially noticeable in classification accuracy, also in better comprehensibility (although more difficult to measure) when compared to the other two groups. At the same time, their efficiency remained similar to those in the second group.

With measurements presented in this paper we give additional arguments for successfulness of multiple knowledge by explicitly measuring the influence of the number of learning examples and the influence of noise, as well as the influence of the error estimate and impurity functions. Benchmarking was performed on two often used domains - lymphography and primary tumor (Clark & Niblett 1989; Michalski et al. 1986; Cestnik & Bratko 1988; Gams 1988).

[1] By 'multiple knowledge' we refer to multiple models, multiple systems or multiple methods.

Here we present results of benchmarking over a period of 7 years. Testing was performed always on the same two oncological domains. Altogether, around 20 systems were benchmarked. Our system GINESYS was constructed on the bases of first benchmarking of around 10 systems in 1987 from a frustration since statistical systems have regularly achieved better accuracy than single AI systems. GINESYS is described in Section 3, benchmarking in Sections 4 and 5.

2 Multiple Knowledge and Multistrategy Learning

Even first expert systems like MYCIN (Shortliffe 1976) and most rule-based systems already enabled a certain amount of multiplicity, i.e. redundancy, since rules can be more or less multiple. Newer systems like CN2 (Clark & Niblett 1989) or C4 (Quinlan 1987) contain similar amount of redundancy which is probably one of the reasons for their successful behaviour in noisy domains. In (Catlett & Jack 1987) it was reported that constructing a separate decision tree for each class with the same method as when constructing one decision tree for all classes significantly increased accuracy. Similar conclusion was derived by Clark & Boswell (1991) when constructing several lists of rules and by Buntine (1989) when combining 5 decision trees with different roots.

In communications, the positive effect of using redundant bits is known for decades and even simple ID numbers in banking have additional digits in order to improve the robustness of the whole system. Theoretical aspects of redundancy in such cases are described e.g. in (Shannon & Weaver 1964).

In most every-day activities, people use multiple knowledge whenever there is any possibility of biasing (Utgoff 1986). For example, when hiring a new employee, one checks several reports which are basically multiple (e.g. biography, recommendations etc.). When bringing an important decision, humans often discuss possibilities in groups of relevant people. A council of physicians is consulted when dealing with difficult or important cases. One physician suffices for most of normal activities since one is substantially cheaper than a group of them.

It is commonly accepted that cross-checking of several knowledge sources is generally better than using one source of knowledge alone. Humans are intrinsically multiple. They apply multiple strategies in every-day activities without paying much attention to that phenomenom. Therefore, machine and human multistrategy learning have natural interrelationship and potential benefits in both directions.

In recent years there were several distinguished events related to mutistrategy learning. Among them: a book edited by R. Michalski & G. Tecuci: *Machine Learning, A Multistrategy Approach*, Vol. IV, Morgan Kaufmann (1994), specialised international workshops on multistrategy learning organised by George Mason University, special issue of Informatica (Tecuci 1993), and IJCAI-93 workshop on integration of machine learning and knowledge acquisition (Tecuci, Kedar & Kodratoff 1993).

3 GINESYS

GINESYS (Generic INductive Expert SYstem Shell) is one of the oldest systems actively utilising multiple knowledge representations (Gams 1988). It consists of two systems (i.e. methods), one from AI and one from statistics. There were sensible reasons for combining methods from different fields. First of all, artificial intelligence methods enable construction of knowledge bases which are typically very transparent and understandable; therefore, it was hoped that a combination would still be more understandable than statistical knowledge bases. A statistical method was chosen on the basis of the hypothesis that knowledge representations should be as different as possible.

GINESYS utilises two different strategies on the basis of these two systems: the AI system constructs and consults lists of rules, and the statistical system multiplies probabilities according to the distribution of classes corresponding to each attribute of the tested example. Both single systems already implicitly utilise multiple knowledge – the AI part through a couple (typically 5) of rules attached to the main one which are triggered when classifying, and the statistical part through combining probabilities relating to the value of each attribute of the tested example.

The AI part of GINESYS is named INESYS

```
d := (()); (*d is initialised*)
repeat
  Star := (NP); BestRules := (NP);
  repeat
    for all Rule_j from Star generate all specialisations
    NewRule_i that do not fulfil the stopping criteria;
    Star := ();
    put into Star at best MAXSTAR the best NewRule_i
    evaluated by user defined impurity function;
    from rules in BestRules and significant NewRule_i choose
    the best MAXBEST rules, evaluated by user defined
    error estimate function, and put them into BestRules
  until Star is empty;
  add BestRules into d;
  L := L - examples, covered by the best evaluated rule from
  BestRules
until L is empty
```

The INESYS algorithm

(see top of the page). It reimplements many of mechanisms of the ID3 and AQ (CN2) family of algorithms. It was primarily designed as an attempt to fully simulate the family of ID3 and AQ inductive empirical learning systems (Gams & Lavrač 1987). Theoretically, it simulates N^M different algorithms where M is the number of modules of the algorithm and N is the number of variations of each module (Gams 1989). The actual number of different variations of GINESYS can be estimated to several hundreds.

INESYS constructs rules with a beam search over all possible combinations of attributes. In addition, it utilises several search-guiding and error-estimate functions such as informativity, the Gini index, Laplacean error estimate and significance. Due to elaborate mechanisms for noise handling, INESYS typically constructs a small number of short rules, i.e. with a small number of attributes. For example, on average, 5.1 main rules with 1.4 attributes in a rule were constructed in lymphography. In primary tumor, there were 11.0 main rules with 2.3 attributes in a rule. Therefore, a typical rule had the form:

if $(A_i = v_{ij})\&(A_k = v_{kl})$ *then* $Distribution_n$
where

- $(A_i = v_{ij})$ is a Boolean test whether attribute i has value j, and
- $Distribution_n$ is a class probability distribution corresponding to the condition part of the rule, i.e. a complex.

A general description of INESYS is:

```
repeat
  construct Rule(s);
  add Rule(s) to d;
  L := L - ExamplesCoveredByRule
until satisfiable d
```

where L is the set of learning examples, d is constructed knowledge in the form of trees or lists of rules and Rule(s) is one or many branches in a tree or one or many rules in the list of rules. A procedural description of INESYS is presented at the top of the page where d is the constructed knowledge in the form of an ordered list of ordered lists of rules, Star and BestRules are ordered lists of rules and L is the set of learning examples. NP is a rule with an uninstantiated complex and class probability distribution of L.

In INESYS, the main improvement regarding existing rule-based systems are rules attached to the main rule. The aim of these multiple rules is twofold. First, to give the user more rules and thus more opportunities to analyse the laws of the domain. Second, to improve classification accuracy by cross-checking the matched rule with con-

firmation rules. This mechanism already enables the use of multiple knowledge to a certain degree:

```
if Complex1 then Class1
   (Complex11 then Class11
        ........
    Complex1R then Class1R)
   else if Complex2 then Class2
          (Complex21 then Class21
               ........
           Complex2R then Class2R)
```

Classification in INESYS starts by sequentially checking main rules. When the first main rule matches a new example, corresponding multiple rules that match the new example add their class probability distribution according to the formula for the union of independent events

$$p_{12} = p_1 + (1 - p_1) \times p_2.$$

Probabilities are multiplied by error estimates in order to calibrate the effect of rules with different credibility, and finally normalised. There are two threshold parameters that present a heuristic estimate of the goodness of classification by a rule: the smallest necessary percentage of the majority class (MINACC) and the smallest difference between the percentage of the majority class and the second to majority class (MINDIFF). Each constructed rule in GINESYS has to satisfy both conditions. Parameter MINDIFF additionally affects the classification process in the sense that the class probability distribution of a combined main and confirmation rules must satisfy it.

The second method in GINESYS is the approximation of the Bayesian classifier which assumes independence of attributes. It is often referred to as "naive Bayes"(Good 1950), in this paper also "Bayes". Naive Bayes constructs all possible rules with only one attribute in the complex. Therefore, the form of these rules is:

if $(A_i = v_{ij})$ *then* $Distribution_n$.

The classification schema is as follows: all rules, that match a new example, are taken in consideration. The probability of each class c is computed by the following formula:

$$P(c|A) = P_a(c) \times (P(A_1|c)/P_a(A_1)) \times \ldots$$
$$\times (P(A_v|c)/P_a(A_v)) \quad (Eq.1)$$

where $P(c|A)$ denotes probability of class c given attributes and values A of the tested example, $P_a(c)$ denotes the a priori probability of class c, $P(A_i|c)$ the probability that attribute A_i has the same value as the classified example regarding the class c, $P_a(A_i)$ the same as before, but regardless of class, and v is the number of attributes. By calculating probabilities of all classes by (Eq.1.), a class probability is obtained. Therefore, although naive Bayes constructs rules similar to INESYS, in the process of classification all attributes are considered in Bayes and on average only around two in INESYS.

Cooperation between the AI and the statistical system is relevant only when they propose different classes. In that case, the goodness of triggered rules in INESYS is estimated by the simple heuristics mentioned above. If the goodness of combined rules exceeds the value of a given threshold (parameter MINDIFF), classification by INESYS is adopted. Otherwise, the classification by naive Bayes prevails. In other words: If class probability distribution of combined rules is estimated as unreliable, the statistical method is called as a supervisor to decide which class is estimated as the most probable.

The combining schema is based on the following reasoning: When multiple rules confirm the main ones, classification is very likely to be correct. If a significant disagreement occurs then the list of rules is not credible and the other method using different knowledge representation should be consulted. It was expected that short rules constructed by INESYS will be more successful when they have high confidence in their prediction, and the approximation of the Bayesian classifier to be more successful when dealing with difficult cases where truncated rules capturing the main and most important laws of the domain are not predicting with great certainty.

4 Benchmarking

Since 1987, systematic measurements are being performed on two oncological domains, lymphography and primary tumor. Data were obtained from real patients from the Oncological institute Ljubljana (Kononenko 1985; Cestnik & Bratko 1988). Unknown values of attributes were re-

SYSTEM	LYMPHOGRAPHY			PRIMARY TUMOR		
	class.acc.	no.rules	no.att.	class.acc.	no.rules	no.att.
GINESYS*	70.5	5.1	7	52.2	11.0	25
GINESYS	70.5	5.1	7	52.0	11.0	25
BAYES	68.6	56.0	56	50.1	37.0	37
CN2-new1	68.7			50.3		
GB*	67.4	5.1	7	47.6	11.0	25
CN2-new1'	65.6			46.9		
NEAREST NEIG.	72.9			40.4		
C4.5-rules	64.7			38.2		
C4.5-trees-u	63.1			48.9		
C4.5-trees-p	66.7			48.8		
CN2-like1	67.3	4.8	8	48.7	11.4	27
CN2-like1'	66.1	5.0	6	45.6	10.8	22
ID3-like	61.8	25.0	110	48.7	28.6	129
CN2-like2	66.8	10.8	21	45.7	19.3	70
CN2-like2'	65.0	9.4	16	46.2	19.4	68
AQ-like1	60.6	7.0	80	48.8	16.0	423
AQ-like2	55.2	7.0	80	32.0	16.0	423

Table 1: Benchmarking systems on two oncological domains.

placed by the most common values regarding the class.

4.1 Domain Description

Basic statistics of the whole set of data are:

LYMPHOGRAPHY
18 attributes
2 - 8 (average 3.3) values per attribute
9 classes
150 examples
distribution: 2 1 12 8 69 53 1 4 0
all examples differ even if one attribute is deleted

PRIMARY TUMOR
17 attributes
2 - 3 (average 2.2) values per attribute
22 classes
339 examples
distribution: 84 20 9 14 39 1 14 6 0 2 28 16 7 24 2 1 10 29 6 2 1 24
75 examples in the data set have another example with the same values of attributes and different class; if we delete one attribute, this number is: 114 111 81 122 84 75 93 79 97 91 77 83 76 77 79 94 94

4.2 Benchmarked Systems

On the benchmark domains, around 20 AI and statistical systems were compared over more than half of a decade. All the systems were given the same set of 10 random distributions of data, each time taking 70% of data for learning and 30% of data for testing. Results of relevant systems are presented in Table 1. The row in the middle of the Table divides multiple and single systems, i.e. those that use only one rule or combine many rules during one classification.

GINESYS* is a version of GINESYS using "negation" multiple rules, which try to confront the main rule if possible. BG* is GINESYS* without the statistical method, i.e. INESYS with functions B and G. First nearest neighbour algorithm classifies with the class of the nearest nei-

LYMPHOGRAPHY			PRIMARY TUMOR		
FUNCTIONS	INESYS**	GINESYS**	FUNCTIONS	INESYS**	GINESYS**
AB	68.4	69.7	BA	48.3	52.3
GB	67.4	69.9	BG	48.1	51.8
BB	66.4	70.8	GB	47.6	52.0
BG	62.6	68.4	AB	46.6	51.3
BA	62.4	68.4	BB	46.4	52.5

Table 2: Accuracy under different impurity and error estimate functions.

ghbour where distance is measured by the number of attributes with different values. BAYES is an approximation of the Bayesian classifier using an assumption that attributes are independent. ID3-like is a version of the ASSISTANT system using cross-validation pruning. CN2-like systems are different modifications of the CN2 algorithm, and CN2-new systems are latest versions. C4.5-rules constructs rules, C4.5-trees-u unpruned trees, and C4.5-trees-p pruned trees. AQ-like systems are modifications of the AQ15 systems.

Classification accuracy (column 1 in each domain in Table 1) was measured as an average percentage of correct classifications in ten test runs. The second column in each domain represents the average number of rules in a rule list or branches in the tree. The third column is a product of the number of rules (branches) times the average length of a rule (branch) times the number of internal disjunctions.

The relations between systems are similar to those observed in other measurements (Clark & Niblett 1989; Rendell et al. 1987; Rendell et al. 1988). Systems of the AQ family usually achieve lower classification accuracy than CN2 or ASSISTANT, while ASSISTANT and CN2 achieve similar classification accuracy. AQ-like1 represents an estimate of the upper possible classification accuracy of the rules, constructed by the AQ-like system. BAYES achieved better results than other systems except GINESYS. Nearest neighbour algorithm seems to be very domain dependent. GINESYS achieved the best average classification accuracy over both domains.

AQ-like systems construct more complex rules than other systems. However, the third column might be misleading for tree constructing algorithms like ID3-like because it represents tree as a list of separated branches. GINESYS* and GINESYS are measured only by the main rules and not by the multiple ones. On the other side, from the results in Table 1 it follows that systems like GINESYS and CN2 construct smaller number of shorter main rules while AQ-like systems construct more complex rules.

The efficiency of the benchmarked algorithms was also analysed. AQ systems are about an order of magnitude slower than ASSISTANT, CN2 and GINESYS, and these are about an order of magnitude slower than BAYES. Results are similar to other measurements when having in mind that our versions of CN2 and GINESYS use a data compression mechanism which speeds up the algorithm roughly five times. GINESYS PC, another version of GINESYS, runs on IBM PC computers and is available as a free scientific software.

4.3 Varying Impurity and Error Estimate Functions

In order to verify whether improvements in GINESYS were caused by multiple knowledge or by domain-dependent parameters, several parameters were varied, and functions were the first among them. GINESYS uses two different groups of functions: informativity functions and error estimate functions. Informativity functions strategically guide search by trying to determine the amount of impurity. Error estimate functions try to estimate classification error. Four functions were used in all 16 possible combinations in each domain. Classification accuracy of GI-

LYMPHOGRAPHY	% OF LEARNING EXAMPLES						
SYSTEM	20%	30%	40%	50%	60%	70%	80%
GINESYS	52.8	58.2	63.1	63.7	60.1	70.5	75.3
BAYES	52.8	59.3	60.8	61.2	58.2	68.6	72.1
INESYS	39.2	51.7	54.1	62.6	59.0	67.4	74.3
ASSISTANT	53.9	60.5	57.9	57.5	55.2	62.1	65.2
ASSIST 0	53.2	60.7	57.4	57.8	55.9	62.4	66.8

Table 3: Accuracy in lymphography at different percentages of learning data.

NESYS with (GINESYS**) and without (INESYS**) top-level multiple knowledge was compared. In Table 2 we present only the best three combinations of INESYS** in both domains. The four functions used were: I - informativity (Quinlan 1986); A - % of majority class; G - Gini index (Breiman et al. 1984); B - Laplacean error estimate (Niblett & Bratko 1986). The first letter denotes the impurity function and the second letter the error estimate function.

Measurements presented in Table 2 indicate that Laplacean error estimate is one of the most successful functions used for impurity or error estimates. Informativity is unexpectedly not present in the best three combinations. Default functions for GINESYS systems (GB) were taken in advance from the literature (Breiman et al. 1984; Niblett & Bratko 1986).

4.4 Varying Percentage of Learning Examples

Benchmarks in sections 4.2 and 4.3 were performed on 10 distributions of data each time taking 70% of data for learning and 30% of data for testing. In Table 3 and 4 we varied the percentage of learning data from 20% to 80% and used the remaining data for testing. Graphical representation of data in Table 4 is shown in Figure 1. Systems in Figure 1 are denoted as in column 1 of Table 4. ASSIST 0 is ASSISTANT without pruning and INESYS is GINESYS without the statistical method.

Probably the main reason for unproportionally low classification accuracy of INESYS with small number of learning examples are functions which work well only with several ten examples. But even then there are some cases when INESYS classifies better than BAYES. The combining mechanism usually decides well when to choose the right method. The performance of INESYS increases with the number of learning examples, and the gain of GINESYS over BAYES also proportionally increases. In lymphography, ASSISTANT prunes the tree by approximately 50% and achieves very similar classification accuracy as ASSIST 0. In primary tumor, the pruned tree constructed by ASSISTANT is roughly 4 times smaller than the tree of ASSIST 0 which besides constructing more complex trees also achieves lower classification accuracy.

The improvement of GINESYS over the best of it's two subparts was typically around 1-2% leading to a conclusion that the combining mechanism performed well when changing the number of learning examples.

4.5 Varying Additional Noise

Noise was introduced into the lymphography and primary tumor domain to attributes and classes in the learning and test examples. For example, 1% of noise means that, on average, each hundred's value of attribute and each hundred's class was randomly changed in learning and test data. Average results of 10 tests (see section 4.2) are presented in Tables 5 and 6, and in Figure 2.

When the amount of noise increases, the performance of INESYS relatively improves and achieves even better classification accuracy that GINESYS. As expected, in a very noisy situation, a small number of short rules performs the best. Si-

P. TUMOR	% OF LEARNING EXAMPLES						
SYSTEM	20%	30%	40%	50%	60%	70%	80%
GINESYS (G)	41.9	44.6	48.1	49.0	48.1	52.0	52.3
BAYES (B)	41.8	45.2	47.5	48.0	47.2	50.1	50.3
INESYS (I)	26.9	35.6	33.8	43.5	41.2	45.9	46.7
ASSISTANT (A)	39.8	43.5	43.5	45.9	44.3	47.9	49.2
ASSIST 0 (A0)	39.6	41.6	39.9	41.1	39.6	41.3	41.7

Table 4: As in Table 3, but for the primary tumor domain.

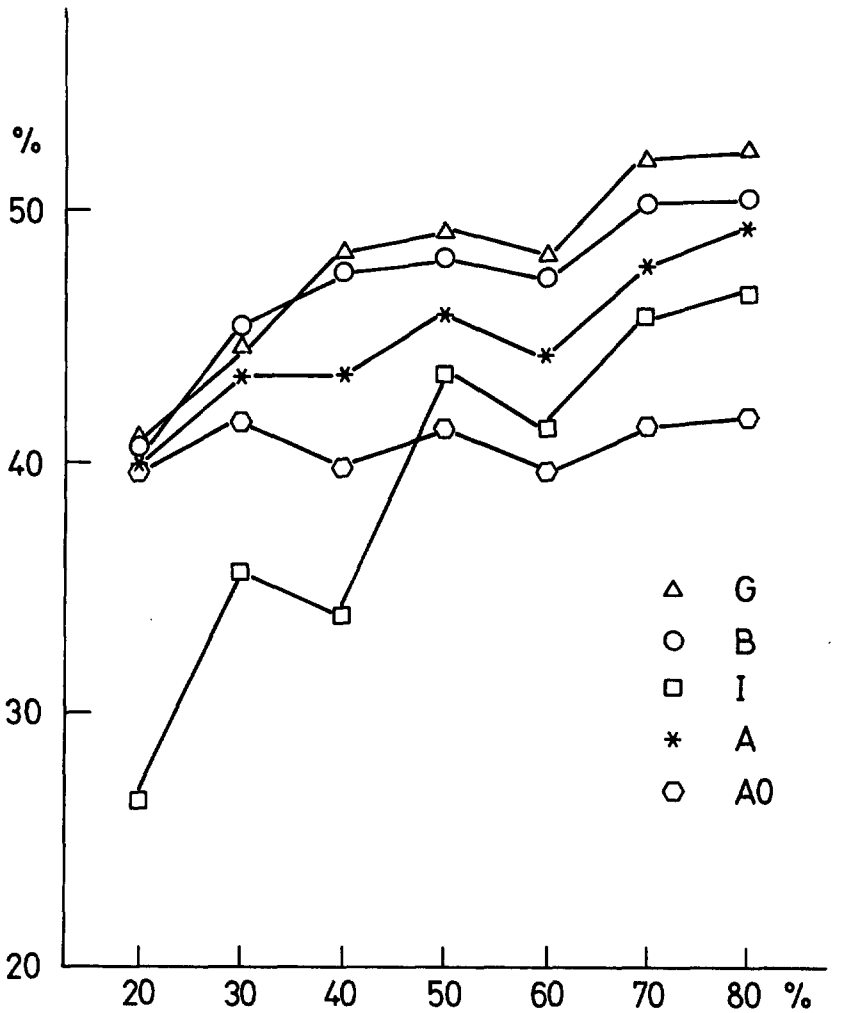


Figure 1: Graphical representation of data in Table 4. On the x-axis is the percentage of learning data and on the y-axis is classification accuracy.

LYMPHOGRAPHY	% OF ADDITIONAL NOISE						
SYSTEM	0%	1%	5%	10%	20%	35%	50%
GINESYS	70.5	65.3	63.7	53.1	43.8	28.9	21.1
BAYES	68.6	65.8	61.7	51.1	41.8	28.0	20.7
INESYS	67.4	63.4	59.1	53.0	41.4	30.3	25.4
ASSISTANT	62.1	60.2	52.8	34.1	33.3	23.4	18.4
ASSIST 0	62.4	60.5	51.8	41.6	29.9	23.5	17.6

Table 5: The influence of additional noise – lymphography.

P. TUMOR	% OF ADDITIONAL NOISE					
SYSTEM	0%	1%	5%	10%	20%	35%
GINESYS (G)	52.0	50.6	42.6	35.2	23.5	13.8
BAYES (B)	50.1	47.8	40.3	33.5	23.6	13.9
INESYS (I)	45.9	43.5	36.2	30.7	20.0	16.1
ASSISTANT (A)	47.9	44.9	39.4	30.5	16.7	8.4
ASSIST 0 (A0)	41.3	39.1	32.4	25.3	14.5	8.7

Table 6: The influence of additional noise – primary tumor.

milar effect is noticeable in the lymphography domain especially compared to ASSISTANT and is probably connected to the fact that ASSISTANT constructs a tree of several tens of leaves while INESYS constructs from 2 to 5 rules. With a growing amount of noise, the gain of GINESYS slowly decreases but remains around 2% as long as any rule of INESYS can be trusted as the meaningful one.

5 New Measurements

In further attempts to verify the obtained results presented in Section 4, GINESYS and benchmark data were around five years ago sent to over 50 laboratories and declared to be freely available for scientific purposes. The obtained answers can be clustered into two groups: several laboratories benchmarked systems on the proposed two domains, or at least approved the approach. On the other hand, there were some researchers who considered proposed benchmarking of classification accuracy as a numerical measurement belonging to statistics. In their opinion, artificial intelligence methods should be evaluated mainly at the level of ideas. Indeed, measuring only classification accuracy does not consider several important advantages of artificial intelligence, e.g., the transparency of the constructed knowledge base or the comprehensibility of classifications. However, in the last two years we have observed a constant shift in a direction which accepts such verifications as crucial in evaluating quality.

In 1990 we received the first, and so far only report of a system, NAIVE BAYES* (Cestnik 1992), which achieved better accuracy than GINESYS in both domains (54.1% in primary tumor and 70.9% in lymphography). The improvement is based on a correction of the weakness of NAIVE BAYES which happens whenever there is a gap in the data, meaning there is no example with the particular value of the attribute. Then, one factor in the product becomes 0 and the resulting product (Eq.1) becomes 0. This was already

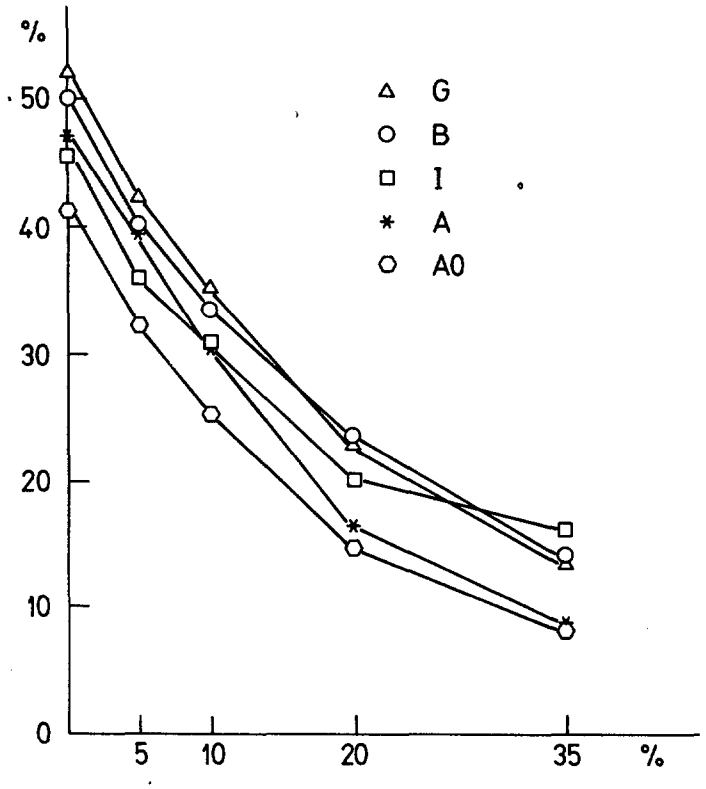


Figure 2: Graphical representation of data in Table 6. On the x-axis is the percentage of additional noise and on the y-axis is classification accuracy.

observed in (Gams & Drobnič 1988; Gams et al. 1991) where ϵ was used instead of 0. In NAIVE BAYES*, the Laplacean estimate is introduced for a correction instead of ϵ.

The reported improvements enabled additional experiments in trying to construct a multiple system, achieving even better classification accuracy. In the first attempt, NAIVE BAYES* was directly embedded into GINESYS, but the observed classification accuracy was lower than that of NAIVE BAYES*. Obviously, a smaller number of stronger rules had to be constructed since NAIVE BAYES* achieved significantly better classification accuracy than GB. Several parameters in GINESYS deal with rules, such as significance (Kalbfleish 1979), modified Laplacean error estimate (Niblett & Bratko 1986) or MINDIFF and MINACC. In the second attempt, MINDIFF was set to 0.5 instead of the previous 0.3, and MINACC to 0.7. Consequently, GINESYS90 achieved an additional 0.8% increase in primary tumor and 1.3% in lymphography over NAIVE BAYES*. Later it was found that the values of MINACC and MINDIFF belong to the set of optimal combinations, as can be observed in Tables 8 and 9.

The updated versions of NAIVE BAYES and GINESYS achieve the best two classification accuracies (compare Table 1 and Table 7). The percentage of corrections by NAIVE BAYES was 8% in lymphography and 27% in primary tumor in GINESYS and, correspondingly, 25% and 45% in GINESYS90.

New values of parameters MINDIFF and MINACC force GINESYS90 to construct a smaller number of longer rules. Also, rules are usually roughly twice more often corrected by NAIVE BAYES* than in GINESYS. To a great extent, this is due to the increased average number of classifications performed by the null or uninstantiated rule, i.e. the last rule in a rule list. This number increased from 9.2 to 15.9 in lymphography (45 classifications), and from 18.0 to 55.1 in primary tumor (102 classifications). Understandably, the last uninstantiated rule is always considered as unreliable in GINESYS and GINESYS90. But in the INESYS and INESYS90 algorithm, the classification is still performed by corresponding null-rule class distribution which is typically only slightly better than the default rule. Therefore, it is understandable that on average accuracy of

SYSTEM	LYMPHOGRAPHY			PRIMARY TUMOR		
	class.acc.	no.rules	no.att.	class.acc.	no.rules	no.att.
INESYS90	63.7	3.8	7	36.3	6.9	19
NAIVE BAYES*	70.9	56.0	56	54.1	37. 0	37
GINESYS90	72.2	59.6	63	54.9	44.3	56

Table 7: Accuracy, number of rules, of all attributes.

	LYMPHOGRAPHY									
	ACC.	.1	.2	.3	.4	.5	.6	.7	.8	.9
.9	71.0	+								
.8	72.0	♡							♡	+
.7	71.6	+								
.6	72.0	♡						♡	+	+
.5	72.2	♡				♡	♡	♡	♡	+
.4	70.2	−			−	−		−	+	
.3	70.7	−		−			−	−	+	+
.2	70.2	−								
.1	68.2	−				−			+	

Table 8: Influence of the goodness criterion, GINESYS90, lymphography.

INESYS90 decreased from 67.4% to 63.7% in lymphography and more, from 45.9% to 36.3% in primary tumor. This should not blur the fact that the effective part of INESYS90 which takes part in classifications of GINESYS90 actually achieves better classification accuracy than INESYS.

The influence of the MINDIFF and the MINACC parameters on the classification accuracy of GINESYS90 was further measured, and it was found that there is a wide range of possible combinations which enable similar improvements (see Tables 8 and 9).

The x-axis in Tables 8 and 9 corresponds to MINACC and the y-axis corresponds to MINDIFF ranging from 0.1 to 0.9. The second column of classification accuracies in each Table represents accuracy with current MINDIFF and MINACC <= MINDIFF. Each mark in Tables 8 and 9 represents one ten-runs measurement as follows (in percents):

− bellow 70.9 in lymphography, bellow 54.1 in primary tumor

\+ between 70.9 and 71.9, between 54.1 and 54.6 correspondingly and,

♡ over 71.9 (+1) in lymphography and over 54.6 (+0.5) in primary tumor.

Top-level or global multiplicity in any version of GINESYS can be estimated by the percentage of different classifications of both single systems. In Table 10, it is presented for GINESYS90 in both domains with MINDIFF = 0.3 and 0.5 (MINDIFF = MINACC) on training and testing examples.

Let us measure the internal multiplicity of each monostrategy system. INESYS90 constructs a list of sublists of rules. However, the order of rules is important and the confirmation rules are attached to the main rules. Therefore, each sublist of rules corresponds to a particular subset of train-

	PRIMARY TUMOR									
	ACC.	.1	.2	.3	.4	.5	.6	.7	.8	.9
.9	54.3	+								
.8	54.3	+							+	
.7	54.3	+							+	
.6	54.5	+							+	
.5	54.9	♡				♡	♡	♡	+	
.4	53.5	−			−	−		−	+	
.3	53.6	−		−						
.2	53.1	−								
.1	51.9	−				−		−	+	+

Table 9: As in Table 8, but primary tumor.

MINDIFF	LYMPHOGRAPHY		PRIMARY TUMOR	
	train	test	train	test
0.3	28	26	28	34
0.5	26	29	44	49

Table 10: Percentage of different classifications, i.e. top-level or global multiplicity in GINESYS90.

ing data and there seems to be no natural way to extract many knowledge bases such that each covers the whole measurement space. On the other hand, rules in both NAIVE BAYES and NAIVE BAYES* have the form
$if(A_i = v_{ij})$ *then* $Distribution_n$
and are constructed on the whole training data. Therefore, a list of rules with the same attribute and all possible values of that attribute represents one knowledge base covering the whole measurement space. The average percentage of different classifications of each such knowledge base and the combined knowledge base is presented in Table 11. It should be observed that the same single knowledge bases are used in NAIVE BAYES and NAIVE BAYES*, but they are differently combined. Whatever the case, both NAIVE BAYES and NAIVE BAYES* can be regarded as internally consisting of multiple knowledge bases. Furthermore, these knowledge subbases are quite independent of each other, although they are constructed on the same training data.

Overall, finding a reasonable combination of the two knowledge bases, i.e. GINESYS90, took only one day of work and resulted in achieving an average 1% increase in classification accuracy. The amount of efforts needed was evidently small because only already existing systems had to be modified.

6 Discussion

Multiple knowledge has proven useful in many measurements, first in (Brazdil & Torgo 1990; Buntine 1989; Catlett & Jack 1987; Cestnik & Bratko 1988; Clark & Boswell 1991; Gams 1988; 1989; Gams, Drobnič & Petkovšek 1991), and followed by tens of reports in the last couple of years. In our measurements, classification accuracy of the combined knowledge base was typically better than the accuracy of each single knowledge

SYSTEM	LYMPHOGRAPHY		PRIMARY TUMOR	
	train	test	train	test
NAIVE BAYES	48	50	72	70
NAIVE BAYES*	44	46	65	66

Table 11: Percentage of different classifications in BAYES, i.e. internal multiplicity.

base. However, due to a relatively high standard deviation the statistical significance of this improvement cannot be proved in 10 tests (Gams 1989). On the other hand, additional measurements were performed by varying parameters of GB (form and number of multiple rules, goodness of rules, factor of significance, impurity functions, error estimate functions) and domain parameters (percentage of training and testing data, percentage of additional noise). In this paper we present over 200 measurements each time averaging 10 tests. If we delete measurements with more than 20% of additional noise and those with less than 70 learning examples, we obtain 167 measurements with only 3 cases where (a version of) GINESYS has not achieved the best classification accuracy. The improvement was typically around 1%.

Therefore, the improvement in 167 measurements (each time averaging over 10 tests) is statistically highly significant. Although more intensive measurements were performed in recent years, e.g., (Brazdil et al. 1994), measurements in this paper present one of the longer-lasting efforts.

Besides better classification accuracy, improved explainability and understandability were also reported. Indeed, the informativity of the knowledge base with multiple rules seems to be much better than without them. Multiple rules can be trimmed off and a "usual" knowledge base is obtained as a downgraded version. Since a user can define the number of multiple rules, the preference function and other parameters, it enables a thorough extradition of most valuable rules. The efficiency of the learning algorithms remains practically the same when using multiple knowledge.

In conclusion, more and more indications emerge that "single-knowledge" systems in general do not achieve the performance of "multiple-knowledge" systems. Therefore, multiple knowledge is becoming regularly implemented in recent systems. The reported gains are usually substantial at small additional cost.

While research on monostrategy methods and one-level knowledge representations continues to be of great importance to the machine learning community, the interest and amount of research work in multistrategy learning and multiple knowledge representations rapidly increases over the last couple of years. Expansion is accompanied by great diversification and new approaches.

In general, multiple systems enable greater competence than monostrategy systems relying on one knowledge representation and one computing mechanism. On the other hand, multiple systems demand more understanding of capacities, limitations and cooperation between single systems. Due to the constant growth of computer power, speed and memory requirements have to a great extend diminished, thus bringing the focus to essential research and engineering questions.

ACKNOWLEDGMENTS

This work was supported by the Ministry of Science, Research and Technology, Republic of Slovenia and was carried out as a part of European Project ESPRIT II Basic Research Action number 3095, Project ECOLES. Research facilities were provided by the "Jozef Stefan" Institute. Data were provided by the Oncological institute of the University medical centre in Ljubljana. We are grateful for suggestions from prof. Ivan Bratko.

References

[1] Brazdil P., Gama J. & Henery B. (1994),

"Characterizing the Applicability of Classification Algorithms Using Meta-Level Learning", *Proc. of ECML-94*, Italy.

[2] Brazdil P.B. & Torgo L. (1990) "Knowledge Acquisition via Knowledge Integration", *Proc. of EKAW-90.*

[3] Breiman L., Friedman J.H., Olshen R.A. & Stone C.J. (1984) *Classification and Regression Trees*, Wadsworth International Group.

[4] Buntine W. (1989) "Learning Classification Rules Using Bayes", *Proceedings of the 6th International Workshop on Machine Learning*, Ithaca, New York.

[5] Catlett J. & Jack C. (1987) "Is it Better to Learn Each Class Separately?", Technical report.

[6] Cestnik, B. (1992), *Probability Estimations in Automatic Learning*, Ph.D. Dissertation.

[7] Cestnik B. & Bratko I. (1988) "Learning Redundant Rules in Noisy Domains", *Proc. of ECAI*, Munich.

[8] Clark P. & Boswell R. (1991) "Rule Induction with CN2: Some Recent Improvements", *Proceedings of EWSL-91*, Porto.

[9] Clark P. & Niblett P. (1989) "The CN2 Induction Algorithm", *Machine Learning*, Vol. 3, No. 4, Kluwer Academic Press.

[10] Gams M. (1988) *Unifying Principles in Automatic Learning*, Ph.D. thesis, Ljubljana.

[11] Gams M. (1989) "New Measurements Highlight the Importance of Redundant Knowledge", *Proc. of EWSL-89*, Montpellier.

[12] Gams M. & Drobnič M. (1988) Approaching the Limit of Classification Accuracy, *Informatica*, No. 2.

[13] Gams M., Drobnič M. & Petkovšek M. (1991) "Learning from Examples - a Uniform View", *International Journal of Man-Machine Studies*, Vol. 34, No. 1.

[14] Gams M. & Karalič A. (1989) "New Empirical Learning Mechanisms Perform Significantly Better in Real Life Domains", *Proc. of the International Workshop on Machine Learning*, Ithaca, New York.

[15] Gams M. & Lavrač N. (1987) "Review of Five Empirical Leaning Systems Within a Proposed Schemata", in *Progress in Machine Learning*, (ed. Bratko I., Lavrač N.), Sigma Press.

[16] Good I.J. (1950) *Probability and the Weighting of Evidence*, Charles Griffing & Co. Limited, London.

[17] Harmon P., Maus R. & Morrisey W. (1988) *Expert Systems, Tools and Applications*, John Wiley & sons.

[18] Kalbfleish J. (1979) *Probability and Statistical Inference II*, Springer-Verlag.

[19] Kononenko I. (1985) "ASSISTANT: A System for Inductive Learning", *M.Sc. thesis*, Ljubljana.

[20] Manago M.V. & Kodratoff Y. (1987) "Noise and Knowledge Acquisition", *Proc. of IJCAI*, Milano.

[21] Michalski, R. (1994) "Inferential Theory of Learning: Developing Foundations of Multistrategy Learning", in Michalski, Tecuci (ed.), *Machine Learnine, A Multistrategy Approach*, Vol. IV, Morgan Kaufmann.

[22] Michalski R.S. & Larson J. (1983) "Incremental Generation of VL1 Hypotheses: The Underlaying Methodology and Description of the Program AQ11", Technical Report ISG 83-5, Urbana: University of Illinois.

[23] Michalski R.S., Mozetič I., Hong J. & Lavrač N. (1986) "The Multi-purpose Incremental Learning System AQ15 and its Testing Application to three Medical Domains", *Proc. of AAAI 86*, Philadelphia, USA.

[24] Michalski, R. & Tecuci G. (ed.) (1994) *Machine Learning, A Multistrategy Approach*, Vol. IV, Morgan Kaufmann.

[25] Niblett T. & Bratko I. (1986) "Learning Decision Rules in Noisy Domains", *Expert Systems*, Brighton, UK.

[26] Quinlan J.R. (1983) "Learning Efficient Classification Procedures and Their Application to Chess End Games", in Michalski R.S., Carbonell J.G. & Mitchell T.M. (Eds.), *Machine Learning: an Artificial Intelligence Approach*, Tioga Publishing, Palo Alto, USA.

[27] Quinlan J.R. (1986) "Induction of Decision Trees", *Machine Learning 1*, Kluwer Academic Publishers.

[28] Quinlan J.R. (1987) "Generating Production Rules From Decision Trees", *Proc. of IJCAI*, Milano.

[29] Rendell L., Powell B., Cho H. & Seshu R. (1988) "Improving the Design of Rule-Learning Systems", *Proc. of The 8th International Workshop on Expert Systems & Their Applications*, Avignon, France.

[30] Rendell L., Seshu R. & Tcheng D. (1987) "Layered Concept-Learning and Dynamically-Variable Bias Management", *Proc. of IJCAI*, Milano.

[31] Shannon C.E. & Weaver W. (1964) *The Mathematical Theory of Communications*, Urbana, Illinois, University of Illinois Press.

[32] Shortliffe E.H. (1976) *Computer-Based Medical Consultations: MYCIN*, American Elsevier Publishing.

[33] Tecuci G. (ed.) (1993) "Special Issue: Multistrategy Learning", *Informatica*, 4 (1993).

[34] Tecuci G., Kedar S. & Kodratoff, Y. (ed.) (1993) *Proc. of IJCAI-93 Workshop Machine Learning and Knowledge Acquisition*, France.

[35] Utgoff P.E. (1986) *Machine Learning of Inductive Bias.*, Kluver Academic Publishers.

Object Migration in Temporal Object-oriented Databases

Angelo Montanari and Elisa Peressi
Dipartimento di Matematica e Informatica, Università di Udine
Via Zanon, 6 - 33100 Udine, Italy
e-mail: [montana,peressi]@dimi.uniud.it
AND
Barbara Pernici
Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza Leonardo da Vinci, 32 - 20133 Milano, Italy
e-mail: pernici@elet.polimi.it

Keywords: object-oriented databases, temporal databases, query languages, roles

Edited by: Rudi Murn
Received: March 10, 1994 **Revised:** November 7, 1994 **Accepted:** November 15, 1994

The paper presents T-ORM (Temporal Objects with Roles Model), an object-oriented data model based on the concepts of class and role. In order to represent the evolution of real-world entities, T-ORM allows objects to change state, roles and class in their lifetime. In particular, it handles structural and behavioral changes that occur in objects when they migrate from a given class to another. First, the paper introduces the basic features of the T-ORM data model, emphasizing those related to object migration. Then, it presents the query and manipulation languages associated with T-ORM, focusing on the treatment of the temporal aspects of object evolution.

1 Introduction

Since the '70s, relational databases have been successfully used in many application domains. In the last years, however, many advanced application areas have been identified for which the data model underlying relational databases is not the most appropriate one. A number of applications in the areas of CAD/CAM, office automation, knowledge representation, software engineering indeed require semantically richer data models. New constructs are needed to model structured entities, complex attribute domains, different types of relationships among entities, relationships among entity types. To support such features, object-oriented databases have been developed, and several commercial systems based on the object-oriented paradigm are now available. In addition, many of these applications must cope with problems involving temporal information about object evolution. For this reason, conventional snapshot databases, that maintain information about the current state of the world only, need to be replaced by temporal databases that record information about past, present and, possibly, future states. In order to model the evolution of real-world entities, object-oriented database must be able to handle both changes in object states and changes in object structures due to their migration to other classes [30]. As an example, they must be able to represent the fact that a person becomes an adult (class change), that a student becomes a professor (role change), and that a student moves from a given university to another one (state change) in a uniform framework.

During the last fifteen years, several time models have been proposed to manage temporal knowledge in database systems. Most of them extend the relational model with one or more time dimensions, e.g. [13, 28]. Temporal extensions of object-oriented models have been proposed in [6, 7, 8, 29, 31]. Most extensions are only concerned with the representation of state evolution,

and neither support the notion of object role, nor allow the shift of an object from one class to another (they assign a class to an object once and for all).

In the paper we consider the problem of providing temporal object-oriented databases with the notion of *object migration*. The importance of such a notion has been pointed out by [20, 30]. In object-oriented databases objects belong to hierarchically structured classes, and remain statically linked to their original position in the hierarchy. On the contrary, in many application domains it is quite natural to allow objects to dynamically change the class(es) they belong to. For example, it seems fairly acceptable to allow an object belonging to the class *PERSON* to *migrate* to the subclass *ADULT*. Only few papers deal with object migration in object-oriented databases using either behavioral constructs to describe semantic information [4], or dynamic integrity constraints [30], or considering a restricted notion of migration [10, 11]. The issue of object migration has not been addressed at all in the context of temporal object-oriented databases.

The paper describes T-ORM [19], a temporal extension of the object-oriented conceptual model ORM (Object with Roles Model [20]), that generalizes the temporal models proposed in [6, 8, 27] by adding the notion of object migration. It first analyzes in detail the notion of object evolution, and consider different types of evolution which can be of interest for database applications; then, it identifies the basic requirements that a temporal model of object evolution must satisfy; finally, it shows how object evolution may affect state, structure and behavior of the evolving object and of the objects related to it. All the features of the T-ORM model are illustrated in terms of query and manipulation languages. There is plenty of literature on temporal extensions to query languages [7, 21, 22, 24, 25, 28, 29, 31] and we do not deviate from them defining a query language which is based on the SQL syntax. Besides the usual primitives of structured query languages, it is provided with all temporal relations of Allen's interval logic and with some specific constructs that allow one to query the history of objects.

The organization of the paper is as follows. Section 2 first illustrates the ORM model and then describes the basic features of its temporal extension T-ORM. Section 3 introduces and discusses the notion of object migration, and shows how it is dealt with in T-ORM. Section 4 provides a detailed presentation of the T-ORM query language. Section 5 sketches out the basic features of T-ORM data manipulation language. The concluding remarks provide an assessment of the work.

2 The T-ORM data model

2.1 Classes and roles

One of the main problems in real-world modeling is the management of object behavior. Most of the efforts in this area have been limited by static schema definitions, supplying objects with methods which operate on object states. Recently, it has been suggested to incorporate rules within objects for expressing object behaviors. Besides the necessity of representing changes in state, another problem occurs. Many applications have the necessity of describing particular entities from different perspectives, dealing with multifaceted object states, that is, an object can play different *roles* and its behavior depends on the role it plays. The term role has been used in various contexts with different meanings. As regards our approach, similar concepts have been developed by Richardson and Schwarz [23], Su [30], Wieringa [32], Sciore [26], and Papazoglou [18].

The ORM model has been originally proposed as an object-oriented design framework for specifying information systems requirements. Such a model allows one to represent object behavior by means of the concept of *role*. A role is a state in which an object can be and if the object is in that state, we say that it *plays* that role. Traditional class-based object-oriented systems model the various states which an entity may assume using specialization hierarchies and representing real-world entities as instances of the most specific class they belong to. This approach has numerous drawbacks. Consider the following example appeared in [32].

> Assume that passenger is a subclass of person and consider a person who migrates to the passenger subclass of person, say by entering a bus. This bus may

> carry 4000 passengers in one week, but counted differently, it may carry 1000 persons in the same week. So counting persons differs from counting passengers.

The conclusion of this observation can be stated in terms of identifiers. If PASSENGER would be a subclass of PERSON, then each passenger identifier would also be a person identifier. Since this is not the case, persons and passengers apparently have different identifiers. We should have a different way to represent those instances. We must realize that a passenger is not *identical* to a person, but that it is a *state* of a person, or, more properly, it is a role of the class PERSON. So, when we count passengers, we really count how often persons have been playing the role of passenger. Moreover, using only the mechanism of class specialization, when we have an entity which can assume different roles independently (for example a person may be a student and an employee), we have to define a separate class which is a subclass of both EMPLOYEE and STUDENT classes. Subclasses of this type should be defined for every possible combination of roles.

In ORM, an object assumes a certain role via a mechanism of instantiation which is analogous to that used to populate classes. We talk about role instances in the same way in which we speak about class instances. Every time that a role is instantiated, we associate a unique identifier (Role Identifier o RID) with the instance which preserves instance identity across changes of its state (i.e., changes to attribute values). We assume that this identifier is unique across the database. All instances of roles evolve independently.

From another point of view, the reason why roles cannot be implemented as subclasses is that the classification mechanism does not allow multiple instantiation. As we said, a person could become a passenger more then once during a week. We cannot instantiate the same person as a passenger more than once and also we cannot think of representing all different kinds of passengers as different subclasses. On the contrary, the role mechanism allows an object to play different roles at different times, to play more than one role at the same time, and to have more than one instance of the same role at the same time (for example, a person who is employed in two different firms). This capability is one of the features that distinguishes the ORM model from other object-oriented models with roles. As an example, in the model proposed in [18] an entity can play several roles simultaneously, but only a single occurrence of each role type is permitted per entity.

At a first glance, one could object that roles represent only particular states which an entity could assume during its lifetime and, as such, one could implement them as a multi-valued time-stamped attribute "state". In general, this is not possible because an object playing a role has a particular behavior specific of that role, which is specified in the role component of a class description through a set of rules and messages and that could not be represented with the traditional way of modeling classes.

A class in the ORM model is defined by a name Cn and a set of roles R_i, each one representing a different behavior of this object:

$$\texttt{class} = (Cn, R_0, R_1, \ldots, R_n)$$

Each role R_i is a 5-uple:

$$R_i = < Rn_i, P_i, S_i, M_i, Rr_i >$$

consisting of a role name Rn_i, a set of properties P_i of that role (abstract description of object characteristics), a set of abstract states S_i that the object can be at while playing this role, a set of messages M_i that the object can receive and send in this role, and a set of rules Rr_i.

Rules fall into two categories: state transition rules and integrity rules. State transition rules define which messages an object can receive/send in each state and the state changes these messages cause. Integrity rules specify constraints on object evolution. This is another aspect which characterizes roles: we can represent object evolution by means of rules and constraints on those rules [9].

Every class has a *base-role* R_0 that describes the initial characteristics of an instance and the global properties concerning its evolution. These properties are inherited by all the other roles; the messages of the base-role are used to add, delete, suspend and resume instances of roles; the possible states in the base-role are pre-defined (*active* and *suspended*); and the rules define transitions between roles and global constraints for the class. Each property has a property name and a domain. Domains may be simple, composite or complex.

Simple domains are predefined domains (such as string, integer, real, boolean), classes, or roles; composite domains are classes and roles; complex domains are defined as aggregates, sets (unordered collections of objects) or sequences (ordered collections of homogeneous objects) of other domains (simple or complex).

Finally, a class can be a subclass of one or more classes (multiple inheritance) and inherits all roles specified in the parent class(es).

2.2 Adding time to objects

Adding the time dimension to object-oriented systems is required for modeling how the entities and the relationships the object denote may change over time [6]. Often an object is created at a given time and is relevant to a system for only a limited period of time. Furthermore, during their existence, objects may change the values of their attributes, the roles that they play, and even the classes they belong to. Temporal (object-oriented) databases may differ from each other both in the structure of the underlying time domain and in the way of associating time information to database entities.

The basic features of time domains have been precisely identified in the literature. Referring to the classification given in [1], we assume that the T-ORM time domain is bidimensional (both valid time and transaction time are supported) and linear in both dimensions, the valid time axis is unbound in both directions, whereas the transaction time axis is bound in both directions (it spans from database creation until the current instant), and both axes map to integers. Furthermore, the time point is taken as the primitive temporal entity (intervals are defined as a derived concept) and the usual metric on integers is defined to measure distances between time points.

With respect to the association of time with data, object attributes can be partitioned in *time-varying* and *constant* ones [17], depending on the fact that their value may change or not over time. The values of time-varying attributes are usually time-stamped at specific time points or intervals; therefore we do not know their value at a time where there is no a specific entry. Common assumptions about their value in such points fall into three categories: (i) *step-wise constant values*, (ii) *discrete values*, and (iii) *values changing according to a given function of time* (e.g. uniformly changing values) [17]. In cases (i) and (iii), the unknown values can be derived from the stored values using a suitable interpolation function. In case (ii), if there is no a specific entry stored at a given time, the attribute must be considered undefined. A further distinction is concerned with the choice of the data unit to time stamp. Two approaches have been proposed in the literature: attribute versioning [5], and object versioning [1]. In the first case, valid and transaction times are associated with each time-varying attribute; in the second case, they are associated with the whole object, and so to all attributes of that object. Attribute versioning presents several advantages, including the following ones: (i) different properties may be associated with time at different granularities; (ii) some properties are inherently not time-varying, so recording time information for them is useless; (iii) time-varying properties of the same object may change asynchronously over time, so as we have to record all object values when a change occurs, we have to duplicate a lot of information (the values which did not change).

Besides associating time information to attributes, object-oriented temporal databases (OOTDBs) can temporally characterize the existence of objects, that is, they can specify when and how an object exists in the database. In most OOTDBs, the set of time intervals during which an object logically exists in the database is called its *lifespan* [6]. This object lifespan spans from the object creation (the point in time when the database first records any information about it) till its complete termination (i.e., logical deletion). As an object can be member of different classes, an object lifespan is the union of its lifespans in all classes in which it has participated. An object lifespan within a class coincides with the union of the lifespans of its properties as a member of that class. In historical object-oriented databases the notion of "reincarnation" is also supported, because a death of an object is not necessarily terminal [6]. For example, employees can be hired, fired, and subsequently rehired.

In the T-ORM model, time is associated with single attributes, class instances and role instances.

With respect to attributes, we assume that

their values are step-wise constant. Therefore an object attribute identifies a sequence of values, each one associated with a different time interval, which has been called *time sequence* (TS) in the literature [27]. Due to the bidimensionality of time, time sequences are constituted by triples $< attribute\ value, valid - time\ interval, transaction - time\ interval >$. Each time interval is represented by a pair $[s, e)$, where s denotes the starting point of the interval, and e its ending point. The interval is closed at the left and open at the right. We assume that valid time intervals for a given attribute are totally ordered with respect to any given transaction point. Finally, if the attribute value has a complex structure, e.g. an aggregate, a set, or a list, we assume that valid and transaction times can be associated with both the whole structure and each of its components. As an example, suppose that the attribute *address* is defined as the aggregate composed of *street* and *town*. Time sequences for address represent changes of values of either *street* or *town*, or both.

With respect to classes and roles, we associate a time sequence with each class (role) instance to denote the time periods during which it is active. The lifespans of role instances and those of the corresponding objects are linked by specific constraints. An object after being suspended can neither send nor receive messages. Therefore, lifespans of role instances are always contained in the lifespan of the corresponding object. Formally, let o be an object instance of a class C, ρ(C) be a function that maps C to the set of roles its instances can play and $r(o, R)$ be a function that maps an object o to the set of its role instances of the role R. The following constraint must hold:

$$\forall R \in \rho(C)\ \forall r \in r(o, R)$$

$$(r.\text{LIFESPAN}) \subseteq (o.\text{LIFESPAN})$$

If

$$(r.\text{LIFESPAN}) = \{[s_1^r, e_1^r), ..., [s_n^r, e_n^r)\}$$

and

$$(o.\text{LIFESPAN}) = \{[s_1, e_1), ..., [s_m, e_m)\}$$

the given constraint states that

$\forall i = 1, ..., n \exists j \in \{1, ..., m\}$ such that $[s_i^r, e_i^r) \subseteq [s_j, e_j)$

All role instances are deleted when the corresponding object is deleted. When an object is suspended, all the roles it has instantiated are also suspended. Object suspension allows us to represent what has been called in [6] object killing and reincarnation.

Let us introduce now a simple schema that will be used in the rest of the paper as a source of exemplification (see Figure 1). We consider four classes, namely PROJECT, DOCUMENT, PERSON and ADULT, which is a subclass of PERSON. Objects belonging to the class PERSON can play two different roles (Employee and Student), each one characterized by its own properties. Projects are developed by persons playing the role of employee. Each project has associated a set of documents written by the employees who participate in the project.

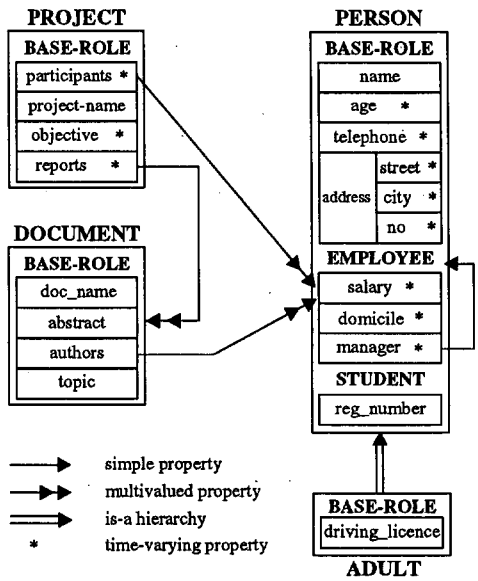


Figure 1: Example of ORM schema

In the following we present some extensions to the ORM model defined in [20] regarding the object-oriented data modeling aspects, and then we explore them in the temporal context. The main concept we examine is object migration. This important issue has still been little researched on. In fact, while existing OODBMSs may capture the notion that an adult is a person, through the mechanism of is-a hierarchies, most of them do not support the notion of a given entity being created as a person and then becoming an adult, that is an entity "migrating" along the class hierarchy it belongs to.

2.3 Composite objects

In object-oriented data models the value of an attribute can itself be an object. In this way, an object can *refer* to another object. In our model we adopt the categorization of references proposed in ORION [15]:

- **weak references**: they are the standard references used in object-oriented systems, and are not provided with any special semantics;
- **composite references** (called also *part-of* relationships): they allow one to define composite objects, i.e., objects composed of other objects.

A composite reference can be:

– **exclusive** or **shared**

In the first case, the referred object can be part of one and only one object; in the second case, it can be part of several composite objects. Two interpretations of exclusivity are possible, depending on its temporal characterization. According to a time-independent interpretation of exclusivity, an object can be part of only one object during its existence. According to a time-dependent interpretation, an object can be part of only one object at each time instant, but it can be part of different objects at different instants. In this second case, exclusivity can be expressed by the following constraint: if an object o is part of the composite objects o' and o'', then the period during which it is part of o' must have an empty intersection with the period during which it is part of o''.

– **dependent** or **independent**

In the first case, the referred object exists (if and) only if the composite object exists, while in the second case, the existence of the referred object does not depend on the existence of the composite object.

The classification of composite references as exclusive or shared, and as dependent or independent are orthogonal, and thus identify four different types composite references.

The main problems involved in the management of composite references concern the relationships between the creation and deletion of composite objects and the creation and deletion of their components. As an example, let o' be a composite object and o be a component of o'. We could state that the deletion of o' causes the deletion of o if one of the following two conditions hold: (i) o' has a dependent exclusive reference to o; (ii) o' has a dependent shared reference to o, but it is the only object currently involved in such a relation with o.

In [6], for instance, a rather restrictive notion of *part-of* relationship is adopted, based on the assumption that a composite object can exist only when its components exist. Such an assumption can be formalized as follows. Let us assume that some composite object o' is defined in terms of n other objects $o_1, .., o_n$ and that its lifespan consists of a set of m disjoint intervals, that is, $o'.LIFESPAN = \{[s_1, e_1), .., [s_m, e_m)\}$. Moreover, let i_m be the number of disjoint intervals belonging to the lifespan of the component object i, for each $i = 1, .., n$. According to the given assumption, the following constraint must always be satisfied:

$$\forall i \; o'.LIFESPAN \subseteq o_i.LIFESPAN$$

which is equivalent to:

$$\forall i, j (1 \leq i \leq n \; \wedge \; 1 \leq j \leq m \; \wedge [s_j, e_j] \in$$

$$o'.LIFESPAN \supset \exists k (1 \leq k \leq i_m \; \wedge$$

$$[s_k, e_k) \in o_i.LIFESPAN) \; \wedge \; [s_j, e_j) \subseteq [s_k, e_k)))$$

Such a solution has two major drawbacks: (i) a composite object cannot be created until all its components have been created; (ii) a composite object must be deleted when one of its components is deleted.

An alternative approach consists in making the existence of a composite object independent from the existence of its components by modeling the *part-of* relationship in terms of roles. This allows us to deal with composite objects which dynamically change their components, supporting the addition/dropping of components to/from a composite object.

3 Migration

In most object-oriented data models proposed in the literature an object is created as an instance of a class with some attribute values and operations associated with it, and remains an instance

of that class till its deletion from the database. This restriction strongly limits the expressiveness of those models. In ORM, it has been partially removed by adding the concept of role, that allows one to deal with the case of an object that plays the same role more than once by the mechanism of multiple role instantiation, preserving the single object identity. As an example, an object of the class PERSON can simultaneously play the roles of Student and Employee (instantiation) and, later, can lose the role of Employee (suspension). The notion of role, however, does not suffice to model the case of an object that migrates from one class to another maintaining its identity (its oid). This means that a member of the class PERSON cannot migrate to the class ADULT maintaining its oid. In the literature, these aspects of object modeling are classified under the general term of *instance evolution.*

3.1 Instance evolution

Instance evolution may assume different forms. In particular, it is possible :

- to let the object migrate to a different class (the object becomes an instance of the new class);
- to specialize the object, that is, it migrates to a subclass (the object becomes an instance of the subclass, but remains a member of the original class);
- to generalize the object, that is, it migrates to a superclass (the object becomes an instance of the superclass and it is no more a member of the original class);
- to dynamically add new classes to an object, so that it can be an instance of more than one class at the same time;
- to dynamically delete classes from an object;
- to specialize or generalize at instance level adding/redefining/deleting attributes and methods for single objects.

These evolutions are controlled by specific semantic constraints in order to restrict the set of classes where an object can migrate to. For example, referring to the schema of Figure 1, a PERSON can become an ADULT, but he/she cannot become a PROJECT. In [33] those constraints are treated as special integrity constraints, which allow one to specify, for each class, its essentiality or its exclusivity. A class C is *essential* if object migration is constrained on the inheritance hierarchy rooted at C. An object could be member of more than one essential class if the model allows multiple inheritance. A class C is *exclusive* with respect to a class C′ if its instances cannot migrate to C′.

In T-ORM we only support two forms of object migration: object generalization and object specialization. In such a way, object migration is allowed only along a unique class hierarchy. This is not an unacceptable restriction if the data model allows the definition of a common root for all class hierarchies. In that case, using an appropriate combination of generalization and specialization operations, we may allow an object to migrate everywhere. In general, however, object migration does not make sense when it occurs between different hierarchies, because it can involve a complete change of the nature and the structure of an object. For example, it does not make sense to allow a person to become a vehicle. One simple way to avoid the problem of unrestricted migrations is to define different class hierarchies (e.g. one rooted on the class PERSON and one rooted on the class VEHICLE) and maintain them separated. The usefulness of having a common root is advocated in [15]. Accordingly, in the ORION system the class hierarchy forms a direct, rooted, acyclic graph (a DAG), having the system-defined class OBJECT as root. That constitutes one of the schema invariants defined by the ORION model in order to maintain schema consistency after schema updating. For example, when we add a new class to the schema hierarchy without specifying its superclass(es), the new class is added as a subclass of the root class OBJECT. It is worth noting that, even in the presence of a common root, one can still avoid object migration between different hierarchies preventing the migration of objects to pass through the root.

In T-ORM, we asssume to have a number of disjoint class hierarchies, that is, T-ORM classes form a disconnected forest and not a tree.

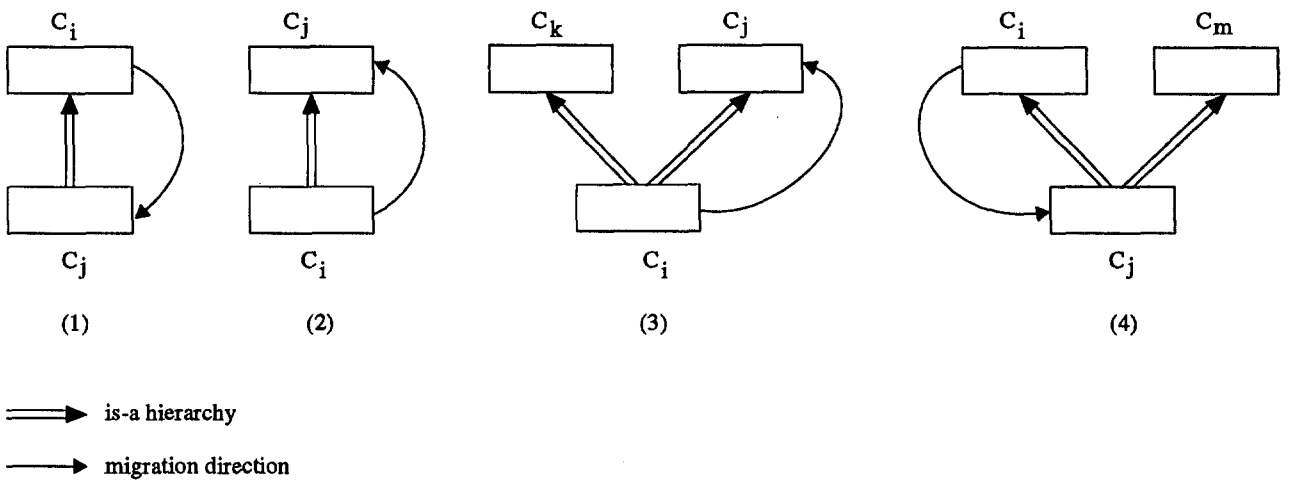


Figure 2: Different cases of object migration

3.2 Constraints on object migration

The inheritance mechanism requires to impose semantic constraints on object migration operators. Let us assume that there is an object o which is an instance of class C_i and the object migrates to class C_j. Consider the four cases illustrated in Figure 2.

case 1) specialization with single inheritance: non-inherited properties defined for class C_j are added to the object; their values are either provided by the user or considered to be null; the object starts its life cycle as a member of class C_j;

case 2) generalization with single inheritance: all properties that are specific for C_i, i.e., not inherited from C_j, are dropped from the object; the lifespan of object o as an instance of class C_i is terminated;

case 3) generalization with multiple inheritance: all properties which are not defined for C_j are dropped; these properties include all properties which are specific for C_i, and all specific or inherited C_k properties not defined for C_j through inheritance links; the lifespan of o as an instance of class C_i is terminated; the lifespans of o as a member of class C_k and its ancestors are terminated appropriately, depending on possible inheritance links between C_k and its ancestors and C_j: lifespans in classes belonging also to C_j ancestors are not terminated;

case 4) specialization with multiple inheritance: all properties of class C_j that are inherited from a superclass C_m of C_j, where C_m is not a superclass of C_i, and all properties specific for C_j are added to the object; their values are either provided by the user or considered to be null. The lifespan of o in classes C_i and all its ancestors, excluded C_i and its ancestors, which are already active, are started.

Consistency of data referring to composite objects has also to be examined in view of object migration. In fact most object-oriented DBMS establish that if an attribute has a class C as domain, its values may be all objects belonging to C or to any subclass of C. If an object o instance of a class C is used as value of an attribute A (with domain C) of an object o′, the migration of o to a superclass of C violates the domain integrity constraint of A. In fact object o′, after the migration of o, will have, as a value of A, an object which is neither instance nor member of A′s domain. We remember that an object is said to be an *instance* of a class C, if C is the most specialized class which the object belongs to. An object is said to be a *member* of a class C if it is an instance of C or of a subclass of C. A solution proposed in [33] allows temporary inconsistency and provides a notification mechanism to determine which objects are inconsistent. In these cases, we adopt the same constraints defined above for the deletion of objects, so that inconsistent references must be dropped.

3.3 Storing information about object life cycle

During its lifetime an object can change roles and migrate along the class hierarchy.

As mentioned in Sect. 2, different kinds of temporal information can be associated to objects:

- The object has associated a lifespan for each instantiated role and for each class of which it is (has been) a member, as discussed in section 2.2. We denote with *oid.LIFESPAN(classname)* the lifespan of *oid* as a member of class *classname* i.e., the set of intervals in which *oid* is instance of the class *classname*. Similarly, we indicate with *oid.LIFESPAN(rolename)* the history of instantiations of role *rolename* for a given object indicated by *oid*.
- The *class-lifespan* stores the history of object migration. It is a time sequence representing the various classes the object is (or was) instance of. The value components are sets of the class types the object belongs to, during the associated valid and transaction time intervals. We indicate with *oid.CLASSLIFESPAN* the time sequence representing migration history for object *oid*.
- The *role-lifespan* is a time sequence which represents the union of the lifespans of the single role instances the object has played during its history. The value components in the time sequence are the sets of role identifiers of the active instances of roles in the associated valid and transaction time intervals. We indicate with *oid.ROLELIFESPAN* the role-lifespan of object *oid*.

The object migration mechanism leads us to impose some temporal constraints on the object lifespan. In particular, if class C_j is an ancestor of class C_i, and *oid* is the identifier of an object which has been member both of C_i and C_j, the following temporal constraint must hold, according to the consistency constraints on migration indicated in the previous section:

$$\text{oid.LIFESPAN}(C_i) \subseteq \text{oid.LIFESPAN}(C_j)$$

Example

Consider the following example of evolution of an object through a series of role instantiations and class migrations (the example is based on the T-ORM schema illustrated in Figure 3 and the history of the object is schematically represented in Figure 4):

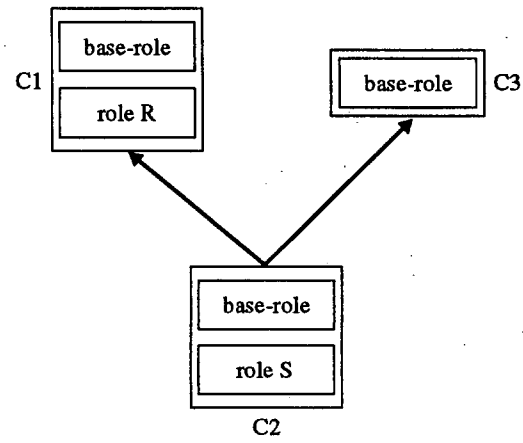


Figure 3: Example of T-ORM schema

- at time t1, the object o1 is created as an instance of the class C_1
- at t2, role R of class C_1 is instantiated the first time as role instance r1
- at t3, role R is instantiated the second time as role instance r2
- at t4, o1 migrates to class C_2
- at t5, role r1 is suspended
- at t6, role r1 is resumed and role S of class C_2 is instantiated as role instance r3
- at t7, role r2 is suspended
- at t8, the object o1 is suspended
- at t9, the object o1 is resumed
- at t10, role r1 is suspended again
- at t11, role r2 is resumed
- at t12, role r1 is resumed

Given the class hierarchy shown in Figure 3, when the object *o1*, instance of class C1, is migrated to class C_2 at time *t4*, its life cycle as an instance of class C_1 continues; in addition, besides starting being an instance of class C_2, it starts also as an instance of class C_3. When the object

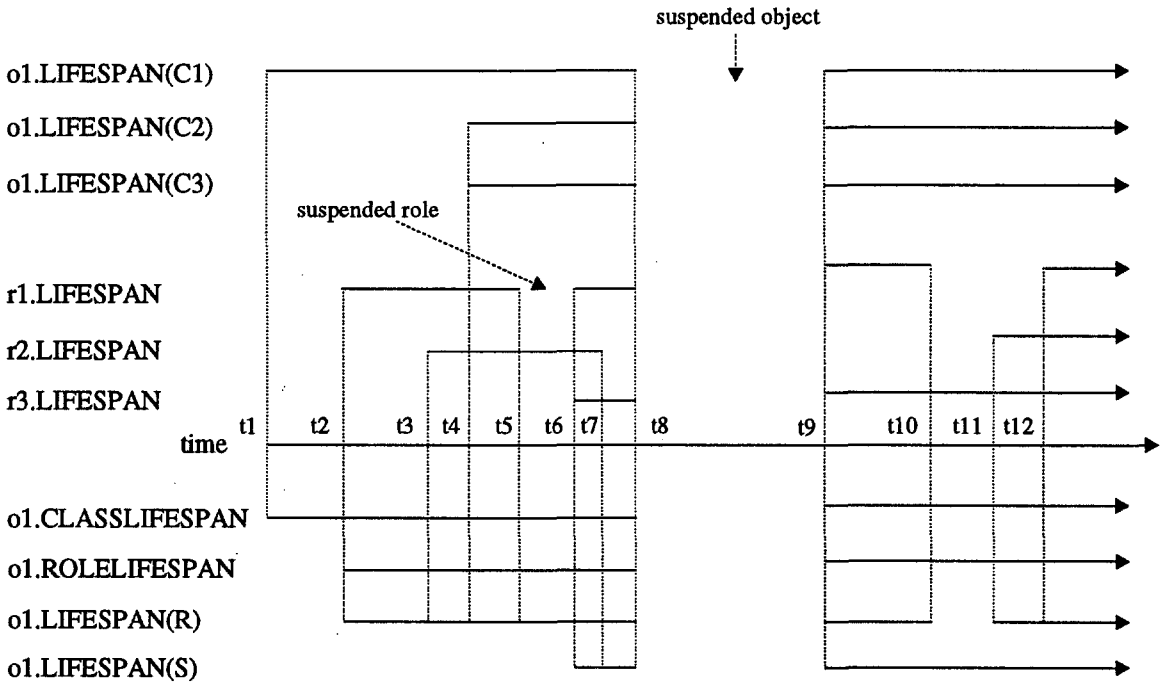


Figure 4: Lifespan dimensions

is suspended at time t8, all active roles are also suspended; roles which were active at the object suspension time are also resumed when the object is resumed.

For the given example, the object lifespan, the role-lifespan, and some of the roles and classes lifespans graphically represented in Figure 4 are shown (for sake of simplicity, only valid times are indicated):

o1.CLASSLIFESPAN = <{C1},[t1,t4)>, <{C1,C2,C3},[t4,t8)>, <{C1,C2,C3},[t9,+∞)>

o1.ROLELIFESPAN = <{r1}, [t2,t3)>, <{r1,r2}, [t3,t5)>, <{r2}, [t5,t6)>, <{r1,r2,r3},[t6,t7)>, <{r1,r3},[t7,t8)>, <{r3}, [t10,t11)>, <{r2,r3}, [t11,t12)>, <{r1,r2,r3, [t12,+∞)>

o1.LIFESPAN(C2) = <[t4,t8),[t9,+∞)>

o1.LIFESPAN(R) = <<{r1},[t2,t3)>, <{r1,r2},[t3,t5)>, ...

r1.LIFESPAN = <[t2,t5), [t6,t8), [t9,t10), [t12,+∞) > <r1,[t6,t8)>, ...

4 Querying T-ORM databases

The complete definition of a data model requires the definition of the corresponding query and data manipulation languages. The goal of querying a temporal database is the retrieval of stored information, taking into account the modifications performed on it. Since bitemporal databases model two temporal dimensions, we can distinguish two basic types of queries: (i) queries that retrieve the sequence of historical values of time-varying information (along the valid time axis); (ii) queries that retrieve data as of a past database state (along the transaction time axis).

In this paper, we focus mainly on queries of the first type that allow us to:

- select an attribute value valid at a given instant, e.g. *find John's salary on 04/15/1986*;
- select an attribute value valid at a time instant associated with another attribute value of the same object, e.g. *find John's salary when Mary was his manager*;
- select an attribute value valid at a time instant associated with another attribute value of another object, e.g. *find John's salary when Mary's salary was $4000*;
- select objects stored in the database during a given time interval, e.g. *find all employees in year 1992*;

– select time intervals starting from attribute values, e.g. *find the time period during which Mary was John's manager.*

General aspects of temporal object-oriented data models require special retrieving properties in order to deal with the concepts of class hierarchy, object identifier, complex domain, complex relationship, valid time, transaction time, time intervals, part-of relationships. Some authors attempted to provide a new query language which is compatible with a relational query language (e.g. SQL in the case of IRIS [12]). Other systems, such as ORION, support a new query language which is based on the nested-relational model. Moreover, there are other features which our query language must take into account introduced by the ORM model, such as the concept of role. Due to those new concepts, we define a language which has suitable operators for additional attribute domains (e.g. time), for all kinds of entity compositions and relationships and which allows selecting a portion of object histories.

Query languages for temporal object-oriented databases, like query languages for conventional databases, are divided in two categories: declarative languages and procedural languages. Declaratives languages allow one to describe a query specifying its target and the conditions it must satisfy, without saying how to obtain the result. Procedural languages, instead, use operators to specify a procedure which tells the system how to obtain the result starting from data. The extensions to existing query languages proposed in literature are based both on declarative languages (such as TQuel [29], extension of Quel, and TOOSQL [25], extension of OSQL) and on procedural languages (e.g. the relational algebra [13]). In an object-oriented perspective it is important to abstract from implementation details, so we think that declaratives languages are the best choice. Thus, the language we define is an extension of the query language of the ORION system [15], and is based on SQL syntax.

In the following we focus our presentation on those aspects which are related to object migration and time In the examples, we refer to the schema of Figure 1.

4.1 Basic query structure

A query has the following structure:

RETRIEVE < `target clause` >
FROM < `specification clause` >
WHERE < `qualification clause` >
AS OF < `as-of clause` >

The *target clause* specifies what parts of the selected information must be retrieved, which could be a set of instances (specifying only an instance variable), a time sequence, a set of values, or a sequence of time intervals (points).

The *specification clause* specifies instance variables used in the query, linking them with the correspondent set of object (role) instances.

The *qualification clause* specifies conditions on time sequences to select particular information. In bitemporal databases we have three dimensions: the data dimension, the valid-time dimension and the transaction-time dimension. The language we are going to define has operators suitable for manipulating all dimensions. In order to maintain the language as simple as possible, we chose to have only one clause (qualification clause) to specify constraints both on the value and the valid-time dimension, whereas other extensions of SQL (such as TSQL [31]) introduce additional clauses.

The *as-of clause* specifies constraints on the transaction-time dimension. It is used to determine the values of object properties as they were recorded sometime in the past and successively revised. In this way we could retrieve information about previous states of the database.

One important element of an object-oriented query language is the facility to express equality between two objects, comparing either their value (value equality) or their oids (object or identity equality). Therefore our query language needs to support both types of equality, which are denoted as $==$ and $=$, respectively.

Because of the nested definitions of objects arising from the class-composition hierarchy, the T-ORM query language must easily allow the specification of predicates on a nested sequence of attributes. In this respect we adopt the well-known *dot notation* to express paths along the class-composition hierarchy (obtaining what we call *path-expressions*).

4.2 Queries on time-varying properties

Time-varying attributes are the main distinguishing characteristics of temporal databases. Each attribute is modeled with a time sequence which represents all its history (see 2.1). A query language must allow the selection of a portion of that history through the specification of conditions either on time, or on attribute values, or both. We can directly select the first two components of < value, valid-time, transaction-time > triplets in time sequences, with the following notations:

```
e.salary.value
e.salary.vtime
```

A path expression of the kind `e.salary` retrieves the time sequence associated with an instance for the specified attribute (salary). We must provide our query language with operators which allow one to select portions of that history. To do that we can use in the where clause predicates with relational operators involving time. Such operators are those of Allen's interval logic [2] (that is PRECEDES, MEETS, OVERLAPS, STARTS, ENDS, INCLUDES, their inverse and EQUAL), those between time points (i.e. <, = and >) and those between time points and intervals (i.e. BEFORE, BEGINS, ENDS, IN, AFTER and their inverse). For example the following query retrieves all values assumed by the property salary of the instance of the class EMPLOYEE whose name is John, which were valid before 04/10/1987. We assume that the property *name* of class PERSON is not time-varying, so it is not modeled as a time sequence, but it assumes only one value.

EX1: *"Find John's salary before that of 04/10/1987"*

```
RETRIEVE s.value
FROM (e,Employee)
WHERE e.name == "John"
AND (04/10/1987 AFTER s.vtime)
```

A path expression which refers to a set of values (such as e.salary) can be quantified using either the existential (EXISTS) or the universal quantifier (FORALL) having the usual meaning. Quantification cannot be made on the variables of the target clause, which are free. We assume that all operators, when applied to a set, distribute on its elements (in the previous example the operator AFTER distributes on the elements identified by s.vtime). Particular elements of a sequence can be selected with the following operators:

- `FIRST(s,e.salary)` $\Longrightarrow$ retrieves the first element in the sequence and assigns it to the variable s
- `CURRENT(s,e.salary)` $\Longrightarrow$ retrieves the current element in the sequence
- `LAST(s,e.salary)` $\Longrightarrow$ retrieves the sequence whose element is the last element in the given sequence
- `<n>-TH(s,e.salary)` $\Longrightarrow$ retrieves the n-th element in the sequence

EX2: *"Find John's current salary"*

```
RETRIEVE c.value
FROM (j,Employee)
WHERE j.name == "John"
AND CURRENT(c,j.salary)
```

Our model is based on time intervals, however we could also select the endpoints of intervals using the functions *BEGIN* and *END* which could be applied to a unique interval or to a sequence of intervals, so they return a single time point or a sequence of time points.

Summarizing, we have defined selection operators on time sequences, which act at different levels of detail, as shown in Figure 5.

4.3 Queries on the history of an object

The other important aspect related to time in object-oriented databases concerns the history of an object as a whole, in addition to considering the history of single attributes as in the previous section.

In our model, we distinguish between *local histories* and *global object histories*. The local history regards single classes and roles and refers to the variations suffered by the set of their instances. The global object history refers to the previous information viewed from the side of the object, that is it contains the history of its variations as member of various classes and instance of various roles.

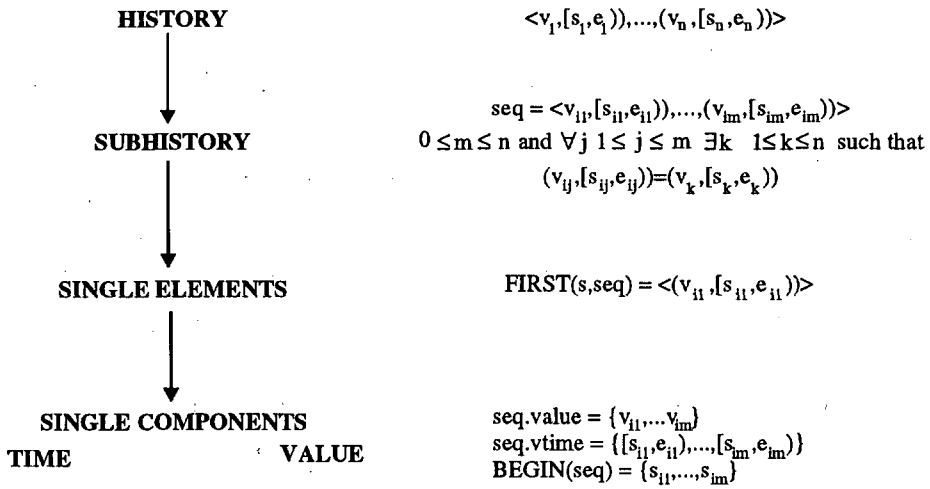


Figure 5: Operators on time sequences

4.3.1 Local history

Due to object migration the set of objects belonging to a class can change over time, and therefore some representational primitives are needed to denote the set of instances of a certain class at a specific time point. They allow a user to query the database to know the time interval(s) during which a given object was an instance of a certain class, to know a particular attribute value or the roles played by the object during such a period of time, or whatever else. The T-ORM query language allows one to refer to instances of a class (role) in three different ways, depending on the considered fragment of the class history:

1. the set of all past and current class (role) instances. According to the previously introduced notation, such a set can be identified as follows :

   (<object-variable>,<class/role name>)

   EX3: *"Find all employees (now and in the past)"*
   RETRIEVE e
   FROM (e,Employee)

2. the set of instances belonging to a given class during a specific time interval. They are retrieved by means of appropriate conditions on the valid time dimension in the where clause:

   EX4: *"Find John's salary in 1975 when he was an employee and his manager was Mary"*
   RETRIEVE s.value
   FROM (e,Employee)
   WHERE e.name == "John"
   AND EXISTS(s,e.salary):
   ([01/01/1975,12/31/1975]) INCLUDES s.vtime
   AND EXISTS(m,e.manager):
   (m.name == "Mary") AND
   ((m.vtime INCLUDES s.vtime) OR
   (m.vtime OVERLAPS s.vtime) OR
   (m.vtime STARTS s.vtime) OR
   (m.vtime ENDS s.vtime) OR
   (m.vtime EQUAL s.vtime))

3. the current set of class (or role) instances

   (<object-variable>, CURRENT(<class/role name>))

   EX5: *"Find all employees (now)"*
   RETRIEVE e
   FROM (e,CURRENT(Employee))

If we denote the set of instances of a class (or role) C at time t with the function o(C)(t), then the following constraint must hold:

$$o \in o(C)(t) \iff t \in o.\mathbf{LIFESPAN}(C)$$

and we have:

(o,**CURRENT** C) returns o(C)(t) with t = NOW

(o,C) returns $\cup_{t\in[-\infty,NOW]}$ o(C)(t)

4.3.2 Migration history

In this paragraph we show how to apply the operators defined for attribute time sequences also to the lifespans time sequences. Remember our representation of object lifespans discussed in paragraph 3.3. We are interested in answering questions of the following type:

EX6: *At which time did Mary become an employee? (role)*
EX7: *At which time did Mary become an adult? (subclass)*
EX8: *During which time period was Mary an employee?*
EX9: *During which time period was Mary an adult?*
EX10: *When did Mary change class?*
EX11: *Which roles did Mary play at 9/6/1994?*
EX12: *Which roles did Mary play during 1994?*

The answer to those questions can be easily found by appropriate queries on the various dimensions of the object lifespan with the use of temporal functions like BEGIN and END. For instance, in query EX10, we select the starting points of valid time intervals from the object class-lifespan, which indicate when a class migration or resuming occurred:

```
EX10: RETRIEVE BEGIN(p.CLASSLIFESPAN.vtime)
      FROM (p,PERSON)
      WHERE p.name == "Mary"
```

In query EX11, we select the role identifiers from the role-lifespan time sequence:

```
EX11: RETRIEVE s.value
      FROM (p,PERSON)
      WHERE p.name == "Mary"
      AND EXISTS(s,p.ROLELIFESPAN):
                (9/6/1994 IN s.vtime)
```

Query EX12 shows an example of using predicates on object history inside the qualification clause:

```
EX12: "Find the salary of the employees who became employees
      before becoming adults"
      RETRIEVE c.value
      FROM (e,Employee)
      WHERE EXISTS
                (s,e.LIFESPAN(Employee)):
                (s.vtime PRECEDES
                FIRST(e.LIFESPAN(ADULT)).vtime)
            AND CURRENT(c,e.salary)
```

5 The Data Manipulation Language

In the DML the operations to create, delete and modify instances have to be extended to involve valid-time specifications. Moreover new operators have to be provided to manipulate the particular features of the extended model such that states, roles and object migration.

5.1 Instance creation

An object is created as an instance of a class. That object could become instance of other subclasses later through the mechanism of object migration. Values for all attributes of that class must be provided by the user, otherwise a null value is assigned by the system. This operation returns the oid of the created object, which can be assigned to an object variable.

If an attribute value is an instance or a set of instances of a class or a role, we can specify those instances directly via their oids (or rids), or indirectly specifying a query. The VALID clause may be omitted. In this case the created object is valid since the time of insertion; an interval whose left end is constituted by transaction time and whose right end is constituted by the value $+\infty$ is inserted in the object lifespan. If only the FROM part is specified, the interval [t1, $+\infty$) is inserted in the object lifespan. Finally if both FROM and TO parts are specified, the interval [t1, t2) is inserted. A time-varying attribute value may be associated with its valid-time. If a validity interval is specified, it must be contained in the validity interval of the whole object. If valid-time specification is omitted, then the attribute value is considered to be valid since time *t1* of the valid clause, or since transaction time if *t1* is not specified, up to time *t2* of the VALID clause, or $+\infty$. If only the FROM part is specified, the attribute value is valid to *t2* or $+\infty$.

In the following example, a PROJECT object is created.

```
EX13: CREATE-OBJECT PROJECT
      WITH (project-name : "P11ts6765",
            participants : {John,Mary} UNION
            RETRIEVE e
            FROM (e,EMPLOYEE)
            WHERE e.manager.name == "Smith",
```

reports : {})

Roles instantiation is similar to class instantiation, but the user must provide the identifier of the object to be instantiated.

In the following example, the Employee role is instantiated for object *John* with the listed attribute values, and starting from Jan. 1, 1993.

EX14: John-empl :
INSTANTIATE-ROLE (John,EMPLOYEE)
WITH (salary : 1600000,
address : "via G. Cesare 57 - ROMA"
manager : %Smith)
VALID FROM 1/1/1993

5.2 Properties updating

Properties updating is an important issue in temporal databases, because we have the possibility of modifying present, past and future data without losing the old one. Updating is not done directly on stored data, but it is performed by insertion of new components in the object history or, more precisely, in their time sequences. Therefore updating an attribute value requires the selection of one or more time sequence components. The selection is based either on the value component, or on the valid-time component, or both. Finally updating existing values requires the "invalidation" of the old ones, and that is done by acting on the transaction-time component.

We could define two different primitives to update and insert information and impose that when we try to update a value during a time period where there is no correspondent time sequence component, the request will be ignored. But in that case, the user should have a precise knowledge of how data are distributed in time. Instead, we chose to have a unique primitive to update and insert information.

The instance to modify is selected via its identifier or retrieving it with the specification of a query on its property values. The temporal qualification of properties can be omitted. In this case the interval [NOW, $+\infty$) is assumed. If the **TO** part is not specified, then $+\infty$ is assumed.

Let us suppose that P1 is a time-varying property whose value is a time sequence like the following:

$$<(v_1,[VT_{i1},VT_{f1}),[TT_{i1},TT_{f1})),$$
$$(v_2,[VT_{i2},VT_{f2}),[TT_{i2},TT_{f2})),\ldots>$$

First of all we must retrieve all time sequence components whose valid-time interval overlaps $[T_{i1},T_{f1})$ and whose transaction-time interval rightend point is $+\infty$ (i.e. the corresponding value is currently valid). These components must be modified as follows according to five cases.
Let $(v_1,[VT_{i1},VT_{f1}),[TT_{i1},TT_{f1}))$ be such a component, we may have the cases illustrated in Figure 6.
We modify the old value for the portion of interval in the object lifespan which overlaps $[T_{i1},T_{f1})$, for the rest of the interval we insert a new component in the time sequence.

Let us follow in detail case a. The specified valid-time interval partially overlaps a valid-time interval in the time sequence. For the portion of time which overlaps with $[VT_{i1},VT_{f1})$ the attribute value must be modified, for the part which does not overlap with $[VT_{i1},VT_{f1})$ a new value is inserted. We must put:
TT_{f1} = NOW (the time sequence component is no more valid)
and insert three new components in the time sequence:

$$(v_1,[VT_{i1},T_{i1}),[NOW,+\infty)),$$
$$(val_1,[T_{i1},VT_{f1}),[NOW,+\infty)),$$
$$(val_1,[VT_{f1},T_{f1}),[NOW,+\infty))$$

As we can note from the figure, after updating we could have two or three contiguous intervals with the same value associated with them. Even if we could collapse the two intervals into one in order to have time sequence components with associated different values, we chose to maintain those intervals separated, because they represent portions of object life which have different histories behind.

The constructs defined for properties updating may be further enriched allowing the specification of operators which calculate the new attribute values starting from the old ones.

5.3 Object migration

In our model, objects can migrate only along the class hierarchy which they belong to, so we consider every class as essential. An object can migrate from a class C to a class C′ which is either a superclass or a subclass of C with the primitives **MIGRATE [UP/DOWN]**.

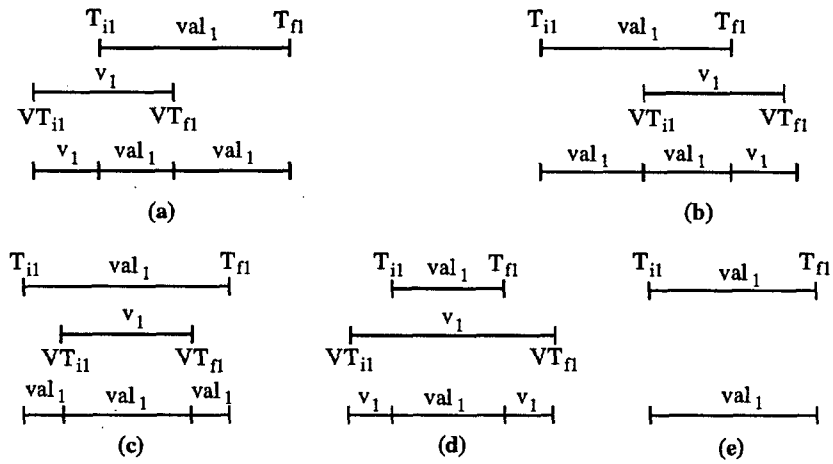


Figure 6: Cases of properties updating

EX15: MIGRATE DOWN (John,PERSON)
TO ADULT
WITH (driving-license : UD56865G9)
VALID FROM 1/1/1993

In this case a new lifespan for the object identified by John as an instance of ADULT starts at time 1/1/1993 and its lifespan as a member of the class PERSONS goes on.
In the case of migration of an object to a superclass, we must check if that causes the violation of domain integrity constraints for some attributes (see par. 3.3) and delete inconsistent references.

We remember also that in order to maintain the identity constraint of objects, object migration does not change object identifiers. Finally in our model we do not consider a hierarchy between roles, such as introduced in [32].

6 Conclusions

Object-oriented data models have several promising features that make them suitable for being extended with new capabilities. In this paper, we studied a temporal extension of an existing object-oriented conceptual model (the ORM model), focusing our attention on object evolution. The basic features of the proposed approach to object migration do not depend on the particular model we chose, and, in principle, can be extended to any other object-oriented data model. The ORM model was chosen for its particular suitability in representing dynamic aspects of object life. We discussed some alternatives for associating temporal information to attributes, to class membership, and to role instantiations. A query and manipulation language have been defined and discussed, focusing on the constructs provided to manage temporal information.

Some remaining open issues concern the definition of a formal semantics for the T-ORM definition, query and manipulation languages, and the generalization of the notion of object evolution to deal with changing schemas. Further work is also needed to model temporal aspects in complex objects, such as variations of object composition in time.

Acknowledgments

This work has been partially supported by the P.A.O.L.A. Consortium (Asem Resolutions, INSIEL, and University of Udine) within the project "Sistemi Multimediali per la Gestione del Patrimonio" and by the Italian Consiglio Nazionale delle Ricerche. The authors would like to thank Nina Edelweiss for her useful suggestions. A preliminary version of this paper appeared in [19].

References

[1] Ahn, I., and R. Snodgrass; A taxonomy of time in databases. SIGMOD Record, Vol. 14, 1985, 236-246.

[2] Allen J. F.; Maintaining knowledge about temporal intervals. Comm. ACM, Vol. 26, No. 11, 1983, 832-843.

[3] Bolour, A, and L.J. Dekeyser; Abstractions in temporal information, Information Systems. Vol. 8, No. 1, 1983, 41-49.

[4] Brodie, M.L.; On modeling behavioral semantics of databases, Proc. Int. Conf. on VLDB, 1981, 32-42.

[5] Clifford, J., and A.U. Tansel; On an algebra for historical relational databases: Two views. ACM SIGMOD 1985, 247-265.

[6] Clifford, J., and A. Croker; Objects in time. IEEE Data Eng., Vol. 11, No. 4, 1988, 11-18.

[7] Dayal, U., and G.T.J. Wuu; Extending existing DBMSs to manage temporal data: an object-oriented approach. In [29].

[8] Edelweiss, N., J.P.M. de Oliveira, and B. Pernici; An object-oriented temporal model. Proc. CAISE 93, Paris, Springer Verlag, June 1993.

[9] Edelweiss, N., J.P.M. de Oliveira, E. Peressi, A. Montanari, and B. Pernici; T-ORM: Temporal aspects in objects and roles. Proc. ORM-1, International Conference on Object-Role Modelling, Townsville, Australia, July 1994, 18-27.

[10] El-Sharkawi, M.E., and Y. Kambayashi; Object migration mechanisms to support updates in object-oriented databases. Proc. PARBASE 1990, 378-387.

[11] El-Sharkawi M.E.; Answering queries in temporal object-oriented databases. Proc. Int. Symposium on Database Systems for Advanced Applications, Tokyo, Japan, April 1991, 21-30.

[12] Fishman, D.H. et al.; Overview of the IRIS DBMS, Chapter 10 in [16], 219-250.

[13] Gadia, S.K.; A homogeneous relational model and query languages for temporal databases. ACM TODS, Vol. 13, No. 4, December 1988, 418-448.

[14] Hartmann, T, G. Saake, R. Jungclaus, P. Hartel, and J. Kusch; Revised version of the modeling language TROLL (TROLL Version 2.0). Tech. Rep. no. 94-03, University of Braunschweig, April 1994.

[15] Kim, W., et al.; Features of the ORION object-oriented database system. Chapter 11 in [16], 251-282.

[16] Kim, W., and F.H. Lochovsky (eds.), *Object-Oriented Concepts, Databases and Applications*, Addison-Wesley, New York, 1989.

[17] Montanari, A., and B. Pernici; Temporal Reasoning. Chapter 21 in [31], 534-562.

[18] Papazoglou, M.P.; Roles: a methodology for representing multifaceted objects. Proc. DEXA 1991, Springer Verlag, 7-12.

[19] Peressi, E., A. Montanari, and B. Pernici; T-ORM: evolving objects and roles, Proc. 4th International Conference on Dynamic Modeling and Information Systems, A. Verbraeck, H.G. Sol, and P.W.G. Bots (eds.), Tech. University Delft, Noordwijkerhout, NL, September 1994, 101-119.

[20] Pernici, B.; Objects with roles. Proc. IEEE/ACM Conference on Office Inf. Syst., Cambridge, MA, 1990, 205-215.

[21] Pissinou, N., K. Makki, and Y. Yesha; Research perspective on time in object databases. In [29].

[22] Pissinou N., Snodgrass R., Elmasri R.; Mumick I., Oszu M.T., Pernici B., Segev A., Theodoulidis B.; Towards an infrastructure for temporal databases - A workshop report, SIGMOD Record, March 1994, 35-52

[23] Richardson, J., and P. Schwartz; Aspects: Extending objects to support multiple, independent roles. Proc. of the ACM SIGMOD Int. Conf. on MOD, Denver, Colorado, May 1991, 298-307.

[24] Rose, E., and A. Segev; A temporal object-oriented algebra and data model. Tech. Rep. LBL-32013, The University of California, Information and Computing Sciences Division, June 1992.

[25] Rose, E., and A. Segev; TOOSQL - A temporal object-oriented query language. Tech. Rep. LBL-33855, The University of California, Information and Computing Sciences Division, March 1993.

[26] Sciore, E.; Object specialization. ACM Trans. on Information Systems, Vol. 7, No. 2, April 1989, 103-122.

[27] Segev, A., and A. Shoshani; Logical modeling of temporal data. Proc. of the ACM SIGMOD Conference, San Francisco, CA, May 1987, 454-466.

[28] Snodgrass, R.; The Temporal Query Language TQuel. ACM TODS, Vol. 12, No. 2, June 1987, 247-298.

[29] Snodgrass, R. (ed.); Proceedings of the International Workshop on an Infrastructure for Temporal Databases. Arlington, Texas, June 1993.

[30] Su, J.; Dynamic constraints and object migration. Proc. of the 16th Int. Conf. on VLDB, September 1991, 233-242.

[31] Tansel, A.U., J. Clifford, S.K. Gadia, S. Jajodia, A. Segev, and R.T. Snodgrass (eds.); *Temporal Databases: Theory, Design, and Implementation.* The Benjamin/Cummings, 1993.

[32] Wieringa, R., and W. de Jonge; The identification of objects and roles - Object identifiers revisited. Technical report IR-267, Vrije University, Amsterdam, December 1991.

[33] Zdonik, S.; Object-oriented type evolution. In Bancilhon, F., and P. Buneman (eds.), *Advances in Database Programming Languages*, Addison-Wesley, 1990, 277-288.

A Novel Approach to Text Compression

H. U. Khan, A. Mahmood and H. A. Fatmi
Dept. of Electronic and Electrical Engineering
King's College London
The Strand, London WC2R 2LS
U.K.
udee795@bay.cc.kcl.ac.uk

Keywords: data compression, Lempel-Ziv coding, text compression

Edited by: Matjaž Gams
Received: March 4,1994 **Revised:** November 8, 1994 **Accepted:** November 17, 1994

A novel algorithm for universal text compression is presented. The proposed algorithm is based on the concept that text compression may be regarded as a pattern recognition problem to which several non-analytical techniques, such as Zadeh's fuzzy theories, could then be applied. The algorithm is compared with the well known Welch algorithm, and results are presented to demonstrate its superiority.

1 Introduction

Lossless data compression is the process of encoding a body of data into a smaller one which can be uniquely decoded back to the original data. Many advantages are obtained by compressing the data, and many techniques have been developed for data compression, which are described in the literature [1, 2]. The two most common applications of data compression are:

(1) Data communication: A sender can compress a data before transmitting it and the receiver can decompress the data after receiving it, thus effectively increasing the data transmission rate of the communication channel.

(2) Data storage: Data is compressed before it is stored and is decompressed when it is retreived, thus increasing the capacity of the storage device.

Amongst the various techniques developed, the most widely used ones include Huffman's arithmetic technique [3], Lempel and Ziv's (LZ) dictionary technique [4], and Storer's textual substitution technique [5]. However, we found that the algorithms are applicable to regions where analytical techniques completely define the compression of sequences and that further improvement can be made by using rule base algorithm [6], based on Zadeh's fuzzy conditional statements [7].

Since Lempel and Ziv's (LZ) pioneering work [4, 8] in universal source coding theory, many attempts have been made to improve upon their ideas and to apply them to data compression and other related fields. Along more practical lines, Welch [9] implemented a modified version of Lempel and Ziv's method [8]. Welch's algorithm works well for many kinds of source data and is being used as a file compression method in several operating systems. However, these techniques have a theoretical limit in Shannon's theorem, which states that the extent to which a message can be compressed and accurately restored is limited by its entropy [10].

This paper can be regarded as a continuation of the improvements to the LZW method for textual data. It is found that the LZW method can be improved by a novel approach to text compression as rule based pattern recognition [6, 11].

The rest of the paper is organized as follows: In section 2, a new algorithm for decoding is presented. A case study is given in section 3 followed by the experimental results in section 4. The paper in concluded in section 5.

2 The proposed algorithm

The new algorithm is organized around the Welch (LZW) dictionary method [9]. This dictionary maps strings of input text into fixed-length co-

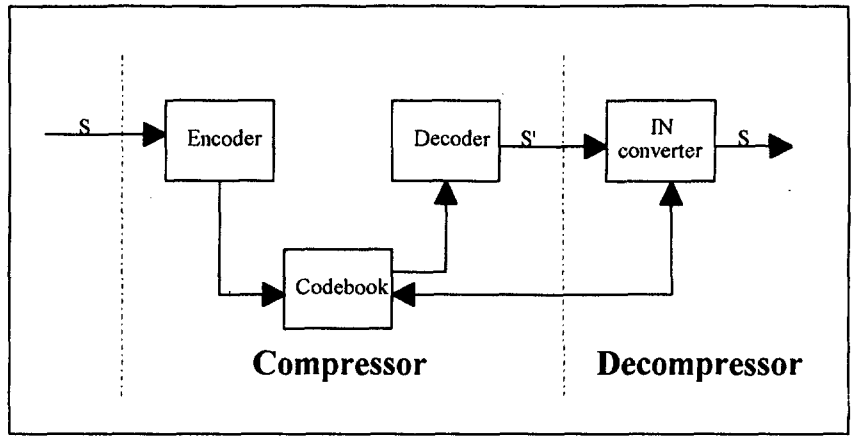


Figure 1: Block diagram for compression and decompression stages

des. The LZW dictionary has a prefix property, i.e., for every string in the dictionary, its prefix string is also in the dictionary. For example, if a string ωC composed of a string ω and a character C and is in the dictionary, then ω is also in the dictionary. The dictionary is initialized to contain all distinguished single characters that appeared in the input text. Each parsed input string is postfixed by the next input character to form a new string. The dictionary is then parsed to search for that string. If a match is found, then the next input character is added to the string and the dictionary is searched again. The input characters are added at the end of the string until a unique string is obtained, which is then added to the dictionary. Each such added string is assigned a unique numeric code. The last added character in the previous string becomes the beginning character for the next string, and the process is repeated to generate the next string to be added in the dictionary.

The proposed algorithm operates on the LZW dictionary to compress the text. The complete algorithm is given below in procedural form. The procedure **Compress** starts decoding from the first two character codes in the dictionary. If two characters are followed by the same number of characters, then it skips the located code and moves to the next code in the sequence. If two characters are followed by more than two character codes, the middle *length(Code[index_no])*-2 characters are selected and their Index Number (IN) is output (using the procedure *Output*). If three or more characters are followed by the same number or more characters, then *length(Code[index_no])*-1 characters are selected. If the code selected is not available in the dictionary, it is split into two parts, and the algorithm recursively searches for the code for the split character strings in the dictionary.

The function of *Code(index_no)* is to return the Character Code (CC) stored against the *index_no* in the dictionary, while the function *index(character_code)* returns the index numbers of the character code from the dictionary.

Various components of the compressor and the decompressor for the proposed method are shown in Figure 1.

```
Procedure Compress
Begin
Input: A text string S
Develop a dictionary on S using LZW algorithm
index_no = 1
While (length (Code (index_no))= 1) Do
      index_no= index_no + 1
Output (Code (index_no))
last_output_index:=index_no
index_no = index_no + 1
While (NOT Endof(dictionary)) Do
  Begin
    If (length (Code(last_output_index)= 2)
       and length (Code(index_no))= 2) Then
     Begin
        index_no = index_no + 1
        Output(Code(index_no))
```

```
    End
  Else
    If length (Code(last_output_index)= 2
       and length (Code(index_no)) > 2)
      Then Begin
        X=Substr(Code(index_no),
           length(Code(index_no) DIV 2),
           length(Code(index_no)) - 2)
        Output (X)
      End
    Else
      If (length(Code(last_output_index)≥ 3
         and length(Code(index_no)) = 2)
         Then Output (Code(index_no))
      Else
        If (length(Code(last_output_index)≥ 3
          and length(Code(index_no))≥ 3)
          Then Begin
            X = Substr(Code(index_no),1,
               length(Code(index_no))- 1)
            Output (X)
          End
    last_output_index = index_no
    index_no = index_no + 1
  End
End
```

```
Procedure Output(X :String)
Begin
   If index (X) in the dictionary Then
      Output_stream = Output_stream +
              index (X)
   Else
      Begin
         Y = Substr (X, 1, length(X) DIV 2)
         Z = Substr (X, length(X) DIV 2
             + 1, length(X))
         Output (Y)
         Output (Z)
      End
End
```

```
Procedure Decompress
Begin
Input: Compressed_string S' Transform S' into S
by replacing INs with respective CCs
output: Original_string S
End
```

The list of CCs generated by the procedure **Decompress** will yield exactly the same string as it was originally fed.

3 A case study

Let us consider the following text string as an example to demonstrate the working of the proposed algorithm.

"Most of the papers published deal with the original work from industrial and Government laboratories, universities and polytechnics."

The dictionary for the above text is given in Table 1. The dictionary is developed by using the LZW encoding method. The symbol ∧ in the dictionary represents a space.

Once the encoding process is completed, it is followed by the decoding of the string. Table 2 shows the selection of codes for decoding the text string by using the proposed algorithm.

The reconstruction of the text starts from the location Row1 and Column1 (R1C1) in Table 2 and ends at the locations R5C11, for this example. The comparison of the new algorithm with the LZW algorithm is given in Table 3.

4 Experimental results

A number of texts with varying lengths were compressed using both the LZW and the new algorithms and the results are shown in the Figures 2,3 and 4.

Figure 2 shows the total number of bytes required by the original texts, the LZW method and the new algorithm. The new algorithm consumed the least number of bytes as compared with the LZW method for all the text strings. This was expected, as the encoding bytes in the new method were less than the LZW method as shown in Table 3. It is also concluded that the Compression Ratio (CR) will be greater as CR is inversely proportional to the decoded numbers as shown in Figure 3. The comparison of percentage compression between the LZW algorithm and the new algorithm is shown in Figure 4. We achieved a maximum of 68.75% compression as compared to 50% compression achieved by the LZW algorithm.

IN	CC	IN	CC	IN	CC	IN	CC	IN	CC	IN	CC	IN	CC
1	M	21	m	41	pa	61	wi	81	mΛ	101	la	121	and
2	o	22	G	42	ap	62	it	82	Λi	102	ab	122	dΛp
3	s	23	v	43	pe	63	thΛ	83	ind	103	bo	123	po
4	t	24	,	44	er	64	Λth	84	du	104	ora	124	ol
5	Λ	25	y	45	rs	65	heΛ	85	us	105	at	125	ly
6	f	26	c	46	sΛ	66	Λor	86	str	106	to	126	yt
7	h	27	q	47	Λpu	67	ri	87	ria	107	ori	127	te
8	e	28	.	48	ub	68	ig	88	alΛa	108	ie	128	ec
9	p	29	Mo	49	bl	69	gi	89	an	109	es	129	ch
10	a	30	os	50	li	70	in	90	nd	110	s,	130	hn
11	r	31	st	51	is	71	na	91	dΛG	111	,Λ	131	nic
12	u	32	tΛ	52	sh	72	alΛ	92	Go	112	Λu	132	cs
13	b	33	Λo	53	hed	73	Λwo	93	ov	113	un	133	s.
14	l	34	of	54	dΛ	74	or	94	ve	114	ni	134	
15	i	35	fΛ	55	Λd	75	rk	95	ern	115	iv	135	
16	d	36	Λt	56	de	76	kΛ	96	nm	116	ver	136	
17	w	37	th	57	ea	77	Λf	97	me	117	rsi	137	
18	g	38	he	58	al	78	fr	98	en	118	iti	138	
19	n	39	eΛ	59	lΛ	79	ro	99	nt	119	ies	139	
20	k	40	Λp	60	Λw	80	om	100	tΛl	120	sΛa	140	

Table 1: The dictionary for the example text using LZW algorithm

R/C	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
R1	29	31	33	35	37	39	41	43	45	47	49	51	53	55
R2	57	59	61	63	4	38	33	67	69	71	14	60	74	76
R3	78	80	82	19	84	86	15	72	89	91	93	95	97	99
R4	5	101	103	11	105	107	109	111	113	115	8	45	62	108
R5	46	89	54	123	125	127	129	131	133					

Table 2: Selected codes for decoding the original text

Text Specification	LZW Alg.	New Alg.
Total characters in the Text String	134	134
Total No. of bits in the Text String	938	938
Largest numeric No. in the dictionary	114	133
Total Codes appeared for the String	107	65
Bits required	749	520
Compression ratio	1.25	1.80
Percentage compression	20.15%	44.56%

Table 3: Comparison of LZW and New algorithms

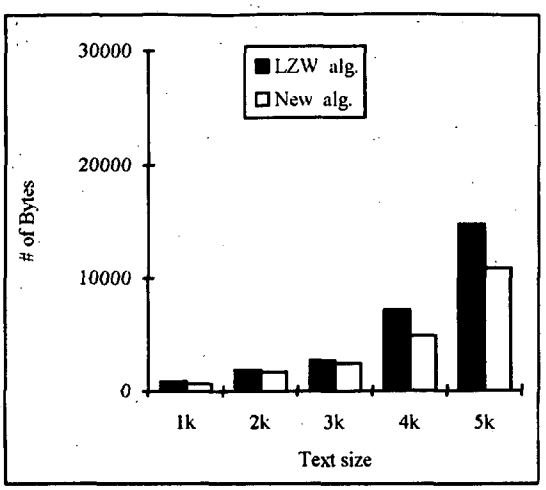


Figure 2: Size of compressed files

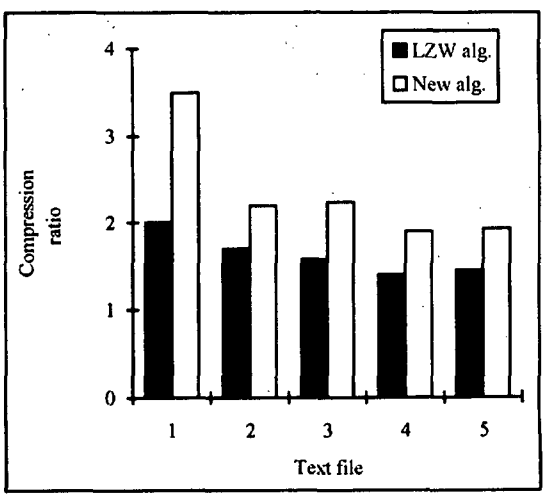


Figure 3: Comparison of compression ratio between LZW algorithm and New algorithm

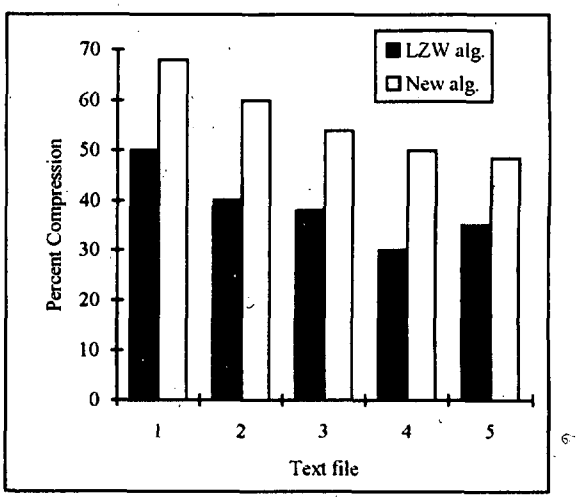


Figure 4: Comparison of Percentage compression between LZW algorithm and New algorithm

5 Conclusions

This paper introduced a novel text compression method which is better than the LZW method.

The experimental results show that a considerable improvement in the compression ratio is obtained when the LZW method is replaced in the decoding part by the proposed non-analytical new algorithm, which removes the redundancy from the LZW algorithm and improves the compression ratio.

Furthermore, the design and implementation of the proposed algorithm is easy to accomplish in terms of both the hardware and the software.

In addition, simulation results on the new algorithm compare favourably with existing methods and are found to be extremely promising. The new algorithm has shown excellent performance with respect to compression ratio capability of text of different sizes.

References

[1] D. Gottlieb, S. A. Hagerth, P. G. H. Lehot and H. S. Rabinowitz: *A classification of compression methods and thier usefulness for a large data processing centre.* in Proc. Nat. Comput. Conf., Vol 44, pp. 453-458, 1975.

[2] T. Bell, I. H. Witten and J. G. Cleary: *Modeling for Text Compression.* Computing Surveys, Vol 21, No 4, pp. 557-591,1989.

[3] D. A. Huffman: *A Method for the Construction of Minimum-Redundancy Codes.* Proceeding of the IRE, Vol 40, No 9, pp. 1098-1101, 1952.

[4] J. Ziv and A. Lempel: *A universal algorithm for sequential data compression.* IEEE Trans. Inform. Theory, Vol IT-23, No 3, pp. 337-343, 1977.

[5] J. A. Storer and T. G. Szymanski: *Data compression via textual substitution.* Jour. ACM, vol 29, No 4, pp. 928-951, 1982.

[6] H. U. Khan, J. Ahmad, A. Mahmood and H. A. Fatmi: *Text compression as rule based pattern recognition.* IEE Electronics Letters, Vol 29, No 20, pp. 1752-1753, 1993.

[7] L. A. Zadeh: *Outline of a new approach to the analysis of a complex system and decision processes.* IEEE Trans. SMC, Vol 3, No 1, pp. 28-44, 1973.

[8] J. Ziv and A. Lempel: *Compression of individual sequences via variable-rate coding.* IEEE Trans. Inform. Theory, Vol IT-24, No 5, pp. 530-536, 1978.

[9] T. A. Welch:*A technique for high-performance data compression.* IEEE Computer, Vol 17, No 6, pp. 8-19, 1984.

[10] C. E. Shannon: *A Mathematical Theory of Communications.* Bell System Tech. Jour., Vol 27, pp. 379-423 and 623-656, 1948.

[11] H. U. Khan and H. A. Fatmi: *Text compression using rule based encoder.* IEE Electronics Letters, Vol 30, No 3, pp. 199-200, 1994.

Recovering 3-D Motion and Structure

Tarek M. Sobh
Department of Computer Science, University of Utah
Salt Lake City, UT 84112, U.S.A.
E-mail: sobh@cs.utah.edu

Keywords: computer vision, robotics, motion recovery, filtering, estimation, image flow

Edited by: Rudi Murn
Received: March 1, 1994 **Revised:** November 17, 1994 **Accepted:** November 24, 1994

We discuss the problem of recovering the 3-D motion and structure. An algorithm for computing the camera motion and the orientation of planar surface is developed. It solves for the 3-D motion and structure iteratively given two successive image frames. We improve the solution by solving the ordinary differential equations which describe the evolution of motion and structure over time. The solution is further improved by exploiting the temporal coherence of 3-D motion. We develop the ordinary differential equations which describe the evolution of the parameters in terms of the current parameters and the measurements. The extended Kalman filter is then used to update the solution of the differential equations. The robustness of the entire process is demonstrated by the experiment with a moving camera which "flies" over a terrain model. This work also examines the possibilities for errors, mistakes and uncertainties in visual sensing systems. We suggest techniques for recovering these 3-D uncertainties, and present examples for determining the parametric evolution of a scene under uncertainty.

1 Introduction

The problem of recovering scene structure and the camera motion relative to the scene has been one of the key problems in computer vision. Many techniques have been developed for the estimation of structure and motion parameters (Tsai and Huang [5], Weng et al. [8] etc.). A lot of existing algorithms depend on evaluating the motion parameters between two successive frames in a sequence. However, recent research on structure and motion has been directed towards using a large number of frames to exploit the history of parametric evolution for a more accurate estimation and noise reduction (Ullman [6], Grzywacz and Hildreth [1], Iu and Wohn [2] etc.)

In this paper we describe a method for recovering the 3-D motion and orientation of a planar surface from an evolving image sequence. The algorithm utilizes the image flow velocities in order to recover the 3-D parameters. First, we develop an algorithm which iteratively improves the solution given two successive image frames. The solution space is divided into three subspaces - the translational motion, the rotational motion and the surface slope. The solution of each subspace is updated by using the current solution of the other two subspaces. The updating process continues until the motion parameters converge, or until no significant improvement is achieved.

Second, we further improve the solution progressively by using a large number of image frames and the ordinary differential equations which describe the evolution of motion and structure over time. Our algorithm uses a weighted average of the expected parameters and the calculated parameters using the 2-frame iterative algorithm as current solution and continues in the same way till the end of the frame sequence. Thus it keeps track of the past history of parametric evolution.

The solution is further improved by exploiting the temporal coherence of 3-D motion. We develop the ordinary differential equations which describe the evolution of motion and structure in terms of the current motion/structure and the

measurements (the 2-D motion vectors) in the image plane. As an initial step we assume that the 3-D motion is piecewise uniform in time. The extended Kalman filter is then used to update the solution of the differential equations. The system was tested on a sequence of images obtained by the motion of a camera over a planar surface.

We also examine the sources of uncertainty in visual sensing systems, and present a method for the recovery of 3-D estimates for motion and structure of an evolving scene. Our method *utilizes* the uncertainty in the 2-D estimates for the image motion to recover the 3-D uncertainty in the *actual* world parameters.

2 3-D Modeling

One can model an arbitrary 3-D motion in terms of stationary-scene/moving-viewer as shown in Figure 1.

The optical flow at the image plane can be related to the 3-D world as indicated by the following pair of equations originally derived by Longuet-Higgins and Prazdny [3], for each point (x, y) in the image plane :

$$v_x = \left\{x\frac{V_Z}{Z} - \frac{V_X}{Z}\right\} + [xy\Omega_X - (1+x^2)\,\Omega_Y + y\Omega_Z]$$

$$v_y = \left\{y\frac{V_Z}{Z} - \frac{V_Y}{Z}\right\} + [(1+y^2)\,\Omega_X - xy\Omega_Y - x\Omega_Z]$$

,

where v_x and v_y are the image velocity at image location (x, y), (V_X, V_Y, V_Z) and $(\Omega_X, \Omega_Y, \Omega_Z)$ are the translational and rotational velocity vectors of the observer, and Z is the unknown distance from the camera to the object.

For planar surfaces, the Z function is simply $pX + qY + Z_o$, where p and q are the planar surface orientations. The situation becomes, for each point, two equations in eight unknowns, namely, the scaled translational velocities V_X/Z_o,V_Y/Z_o and V_Z/Z_o, the rotational velocities Ω_X,Ω_Y and Ω_Z and the orientations p and q. Differential methods could be used to solve those equations by differentiating the flow field and by using approximate methods to find the flow field derivatives. The existing methods for computing the derivatives of the flow field usually do not produce accurate results. Our algorithm uses a discrete method instead, i.e, the vectors at a number of points in the plane is determined and the problem reduces to solving a system of nonlinear equations, a pair of equations represents the flow at each point as follows :

$$v_x = (1 - px - qy)\left(x\frac{V_Z}{Z_o} - \frac{V_X}{Z_o}\right) + [xy\Omega_X - (1+x^2)\,\Omega_Y + y\Omega_Z]$$

$$v_y = (1 - px - qy)\left(y\frac{V_Z}{Z_o} - \frac{V_Y}{Z_o}\right) + [(1+y^2)\,\Omega_X - xy\Omega_Y - x\Omega_Z]$$

It should be noticed that the resulting system of equations is nonlinear, however, it has some linear properties. The rotational part, for example, is totally linear, also, for any combination of two spaces among the rotational, translational and slope spaces, the system becomes linear. For the system of equations to be consistent, we need the flow estimates for at least four points, in which case there will be eight equations in eight unknowns.

3 Two-Frame Algorithm

The algorithm takes as input the estimate of the flow vectors at a number of points ≥ 4 obtained from motion between two images. It iterates updating the solution of each subspace by using the solution of the other two subspaces. Each update involves solving a linear system, thereby it requires to solve three linear systems to complete a single iteration. This process continues until the solution converges, or until no significant improvement is made. The algorithm proceeds as follows :

1. Set p, $q = 0$;
   input the initial estimate for rotation ;
   Solve the linear system for translation;
2. Use the translation and rotation from step 1 ;
   Solve the linear system for the slope ;
3. Set i=1;
   While (i $\leq$ Max. Iterations)
   and (no convergence) Do
   Solve for the rotations using latest
   estimates of translations, p and q;

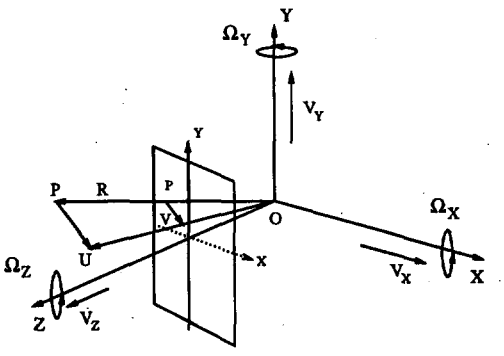


Figure 1: 3-D Formulation for Stationary Scene/Moving Viewer

Solve for the translations using latest estimates of rotations, p and q;
Solve for p, q using latest estimates of translations and rotations;
end While ;

3.1 Complexity Analysis

As we mentioned earlier, one should notice in the equations relating the flow velocities with the slope, rotational and translational velocities that they are "quasi-linear" , if one can say so. The equations exhibit some linear properties. This suggests that a purely iterative technique for solving non-linear equations might not be an excellent choice, since, the variables are linearly related in some way. To think of a way of "inverting" the relations might be a good start, although to do that without a framework based on iterating and gravitating towards a solution is not a good idea.

This makes one think of applying a method which converges faster than a purely iterative scheme like Newton's method. However, the complexity of Newton's method is determined by the complexity of computing the inverse Jacobian, which is of an order of N^3, or $N^{2.81}$ multiplications as the lower bound using Strassen's technique. In our case, since we have at least 8 equations in 8 unknowns, the complexity is of order $8^3 = 512$ multiplications at every iteration, and the method does not make any use of the fact that the set of equations at hand exhibits some linear properties.

The algorithm proposed, on the other hand, makes very good use of the fact that there are some linearity in the equations, by inverting the set of relations for each subspace at every iteration. The complexity at every iteration is of the order of the complexity of computing the pseudo-inverse which is of the order of ($3^3 + 3^3 + 2^3$) multiplications at each iteration, where the first 3 comes from solving the system for the rotational variables, the second 3 is for the translations, the last 2 is for p and q. This is equal to 62 multiplications at every iteration, which is significantly less than the 512 multiplications in a method like Newton's for example. It was noticed that the algorithm converged to solution in a very small number of iterations for most experiments we have conducted so far. The maximum number of iterations was 7.

Using the latest solution obtained from the two-frame analysis as the initial condition for the next two-frame problem in the image sequence would further decrease the complexity, as the next set of parameters would, most probably, be close in values to the current parameters, thus the number of iterations needed to converge to the new solution would decrease significantly.

3.2 Observations

- The algorithm is not sensitive to the initial condition of the orientation parameters. The plane is simply assumed to be a frontal one at the beginning. The slope parameters evolves with iterations.
- The algorithm is sensitive to input noise just like other existing algorithms, some experiments shows the sensitivity with respect to the change of viewing angle, table 5 includes some results of those experiments. Si-

milarly, the algorithm performs better for a large number of points that are evenly distributed throughout the planar surface, than it does for clustered, smaller number of image points.

- It is proven that there exists dual solutions for such systems. However, if our method gravitates towards a "fixed point" in the solution space we can find the other explicitly in terms of the first one from the relations given by Waxman and Ullman [7].

4 Multi-Frame Algorithm

The ordinary differential equations that describe the evolution of motion and structure parameters are used to find the expression for the expected parameter change in terms of the previous parameter estimates. The expected change and the old estimates are then used to predict the current motion and structure parameters.

At time instant t, the planar surface equation is described by

$$Z = pX + qY + Z_o$$

To compute the change in the structure parameters during the time interval dt, we differentiate the above equation to get

$$\frac{dZ}{dt} = p\frac{dX}{dt} + X\frac{dp}{dt} + q\frac{dY}{dt} + Y\frac{dq}{dt} + \frac{dZ_o}{dt}$$

The time derivatives of (X, Y, Z) in the above expression are given by the three components of the vector $-(\mathbf{V} + \mathbf{\Omega} \times \mathbf{R})$ that represent the relative motion of the object with respect to the camera. Substituting these components for the derivatives and the expression $pX + qY + Z_o$ for Z we can get the exact differentials for the slopes and Z_o as

$$dZ_o = Z_o\left[(\Omega_Y + V_X)p - (\Omega_X - V_Y)q - V_Z\right]dt$$

$$dp = \left[p(\Omega_Y p - \Omega_X q) + (\Omega_Y + \Omega_Z q)\right]dt$$

$$dq = \left[q(\Omega_Y p - \Omega_X q) - (\Omega_X + \Omega_Z p)\right]dt$$

Using the above relations, we can compute the new structure parameters at time $t + dt$ as

$$\acute{p} = p + dp\ ,\ \acute{q} = q + dq \ \text{ and } \ \acute{Z}_o = Z_o + dZ_o$$

Thus the slope parameters evolve at time $t+dt$ as follows :

$$\begin{bmatrix} \acute{p} \\ \acute{q} \end{bmatrix} = \begin{bmatrix} p \\ q \end{bmatrix} +$$

$$\begin{bmatrix} \Omega_Y p - \Omega_X q & \Omega_Z & \Omega_Y \\ -\Omega_Z & \Omega_Y p - \Omega_X q & -\Omega_X \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix} dt$$

The new translational velocity $\acute{V}$ at time $t + dt$ can be found in the absence of accelerations from

$$\acute{\mathbf{V}} = \mathbf{V} + \mathbf{V} \times \mathbf{\Omega} dt$$

Dividing $\acute{V}$ by $\acute{Z}_o$ we get the new expected scaled translational velocity components at time $t + dt$ as follows :

$$\begin{bmatrix} \acute{V}_X \\ \acute{V}_Y \\ \acute{V}_Z \end{bmatrix} = \begin{bmatrix} V_X \\ V_Y \\ V_Z \end{bmatrix} +$$

$$\begin{bmatrix} -s & \Omega_Z & \Omega_Y \\ -\Omega_Z & -s & \Omega_X \\ \Omega_Y & -\Omega_X & -s \end{bmatrix} \begin{bmatrix} V_X \\ V_Y \\ V_Z \end{bmatrix} dt,$$

where s is expressed as follows :

$$s = (\Omega_Y + V_X)\,p - (\Omega_X - V_Y)\,q - V_Z$$

The expected rotational parameters at time $t+dt$ remain equal to their values at time t since

$$\acute{\mathbf{\Omega}} = \mathbf{\Omega} + \mathbf{\Omega} \times \mathbf{\Omega} dt = \mathbf{\Omega}$$

and thus

$$\left(\acute{\Omega}_X, \acute{\Omega}_Y, \acute{\Omega}_Z\right) = (\Omega_X, \Omega_Y, \Omega_Z)$$

Our first multi-frame algorithm uses a weighted average of the expected parameters at time $t + dt$ from the above equations and the calculated parameters using the two-frame iterative algorithm as the solution at time $t + dt$, and continues in the same way until the end of the frame sequence. Thus it keeps track of the past history of parametric evolution. We further develop the first multi-frame algorithm to exploit the temporal coherence of 3-D motion. We develop the ordinary

differential equations which describe the evolution of motion and structure in terms of the current motion/structure and the two-dimensional flow vectors in the image plane. We assume that the 3-D motion is piecewise uniform in time, i.e, $\dot{\vec{\Omega}} = \dot{\vec{V}} = \mathbf{0}$. We then use the equations expressing the time derivative of the slope derived above and the fact that the derivative of the rotational velocities is zero and develop the following expressions for the scaled translational velocities and the depth Z_o :

$$\frac{dV_X'}{dt} = -V_X' \frac{1}{Z_o}\frac{dZ_o}{dt}, \quad \frac{dV_Y'}{dt} = -V_Y' \frac{1}{Z_o}\frac{dZ_o}{dt} \quad \text{and} \quad \frac{dV_Z'}{dt} = -V_Z' \frac{1}{Z_o}\frac{dZ_o}{dt}$$

$$\frac{1}{Z_o}\frac{dZ_o}{dt} = -V_Z' - pv_x - qv_y$$

The extended Kalman filter is then used to update the solution of the differential equations. Where the state vector can be written as :

$$X = [\ V_X'\ \ V_Y'\ \ V_Z'\ \ \Omega_X\ \ \Omega_Y\ \ \Omega_Z\ \ p\ \ q\ ]$$

and the measurement vector is expressed as :

$$Z = [\ v_x\ \ v_y\ \ \frac{\delta v_x}{\delta x}\ \ \frac{\delta v_y}{\delta x}\ \ \frac{\delta v_x}{\delta y}\ \ \frac{\delta v_y}{\delta y}\ \ \frac{\delta v_x}{\delta t}\ \ \frac{\delta v_y}{\delta t}\ ]$$

The behavior of the two-frame algorithm and the multi-frame algorithm can be conceptualized as a control system as shown in Figures 2 and 3.

5 Results for the Two–Frame and Multi–Frame Algorithms

The algorithm was run on a sequence of image data. The images were those of a planar surface being approached by a video camera mounted on a robot arm. The plane consisted of 120 dots. The sequence simulated the situation where an airplane approaches a runway for landing. One may think of the dots as lights to guide the airplane during a night landing. The actual rotational and translational velocities between each two subsequent shots were $\Omega_X = -3°$, $\Omega_Y = 0°$, $\Omega_Z = -5°$, $V_X = 0$ mm, $V_Y = 10$ mm and $V_Z = 20$ mm. To determine the flow vectors, the first order moments were used to calculate the center of mass of each one of the dots in the image sequence and then they were matched across the image sequence. Thus, there were 120 points at which the x and y displacements were available as the approximation to the flow velocities. In real image date, more elaborated flow recovery algorithm should be used in order to determine the flow field accurately. Figures 4 and 5 show the image sequence that was used.

Tables 1 and 2 show the recovered and actual parameters when the two-frame algorithm is used. Tables 3 and 4 include the parameters computed from the multi-frame algorithm. Table 5 includes the results of varying the view angle for the two-frame algorithm. It should be noted that some of the parameters improved significantly when we used the filtering mechanism. The translational velocities in the y and z directions and the plane orientation in the y direction were recovered more accurately at the end of the sequence using the second algorithm. However, error propagation caused a slight deterioration in the recovered values of a few parameters at the end of the experiment.

6 Modeling 3-D Uncertainties

In this section we utilize uncertainties in visual sensing to recover 3-D structure and motion characteristics of a scene under observation. The computed uncertainties are used for reconstructing the evolving scene.

Figure 6 depicts the sequence of steps that are to be performed in order to recover the full world uncertainty from 2-D measurements on the image plane. We concentrate on identifying methods by which the 2-D uncertainty could be transformed into meaningful 3-D interpretations that the observer can use reliably in order to recover the world parameters.

7 Recovering 3-D Uncertainties

We suggest the usage of the classical formulation for 3-D parameter recovery from 2-D displacement vectors, but using 2-D error distributions as estimates for motion and/or feature coordinates in order to compute 3-D uncertainty distributions for the real world motion vectors and structure instead of singular values for the world parameters.

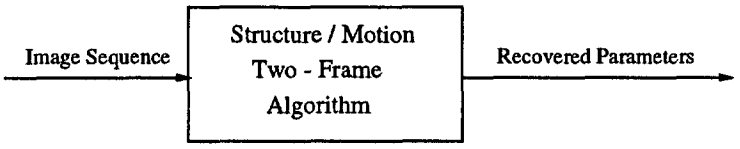


Figure 2: Two Frame Algorithm as a Control System.

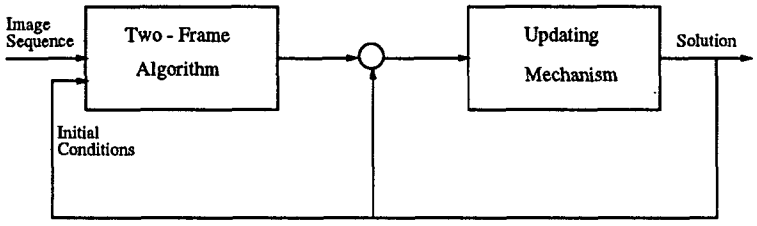


Figure 3: Multi–Frame Algorithm as a Control System.

Image 1 Image 2

Image 3 Image 4

Figure 4: The Image Sequence (Image 1 $\longrightarrow$ Image 4)

Image 5 Image 6

Image 7 Image 8

Figure 5: The Image Sequence (Image 5 $\longrightarrow$ Image 8)

Parameters	*I1 -> I2*	*Actual*	*I2 -> I3*	*Actual*	*I3 -> I4*	*Actual*	*I4 -> I5*	*Actual*
Ω_x	-0.096	-3.0	0.762	-3.0	0.429	-3.0	-0.224	-3.0
Ω_y	-0.162	0.0	-0.042	0.0	0.016	0.0	0.108	0.0
Ω_z	-2.537	-5.0	-2.889	-5.0	-2.82	-5.0	-3.005	-5.0
V_x	1.691	0.0	1.25	0.0	1.276	0.0	1.12	0.0
V_y	9.25	10.0	10.71	10.0	8.22	10.0	4.8	10.0
V_z	15.46	20.0	15.22	20.0	17.55	20.0	18.23	20.0
p	13.14	17.32	4.7	16.56	9.76	14.36	9.05	12.12
q	30.875	12.25	22.95	13.34	29.99	18.56	33.84	22.73

Table 1: Recovered Parameters using the 2-Frame Algorithm (Image 1 $\longrightarrow$ Image 5)

Parameters	*I5 -> I6*	*Actual*	*I6 -> I7*	*Actual*	*I7 -> I8*	*Actual*
Ω_x	-0.136	-3.0	-2.9	-3.0	-3.04	-3.0
Ω_y	0.072	0.0	-3.7	0.0	-0.159	0.0
Ω_z	-2.297	-5.0	-5.32	-5.0	-3.802	-5.0
V_x	0.806	0.0	2.16	0.0	0.7	0.0
V_y	2.963	10.0	5.54	10.0	7.167	10.0
V_z	18.53	20.0	11.35	20.0	16.934	20.0
p	17.94	13.67	22.04	10.23	15.07	4.42
q	13.57	37.38	25.04	45.51	71.06	78.36

Table 2: Recovered Parameters using the 2-Frame Algorithm (Image 6 $\longrightarrow$ Image 8)

Parameters	*I1 -> I2*	*Actual*	*I2 -> I3*	*Actual*	*I3 -> I4*	*Actual*	*I4 -> I5*	*Actual*
Ω_x	-0.1	-3.0	0.29	-3.0	0.358	-3.0	-0.23	-3.0
Ω_y	-0.214	0.0	-0.045	0.0	-0.014	0.0	0.0198	0.0
Ω_z	-2.64	-5.0	-2.87	-5.0	-2.95	-5.0	-3.19	-5.0
V_x	1.634	0.0	1.18	0.0	1.054	0.0	0.765	0.0
V_y	9.933	10.0	10.23	10.0	9.89	10.0	8.45	10.0
V_z	14.49	20.0	15.67	20.0	16.98	20.0	18.045	20.0
p	11.01	17.32	10.07	16.56	11.024	14.36	10.67	12.12
q	26.5	12.25	20.87	13.34	25.88	18.56	28.78	22.73

Table 3: Recovered Parameters using the Multi–Frame Algorithm (Image 1 $\longrightarrow$ Image 5)

Parameters	*I5 -> I6*	*Actual*	*I6 -> I7*	*Actual*	*I7 -> I8*	*Actual*
Ω_x	-0.65	-3.0	-2.83	-3.0	-2.931	-3.0
Ω_y	0.023	0.0	-0.86	0.0	-0.64	0.0
Ω_z	-2.784	-5.0	-4.313	-5.0	-4.157	-5.0
V_x	0.453	0.0	0.729	0.0	0.57	0.0
V_y	6.56	10.0	7.14	10.0	8.85	10.0
V_z	19.032	20.0	17.62	20.0	18.94	20.0
p	14.52	13.67	20.45	10.23	11.293	4.42
q	26.57	37.38	36.47	45.51	75.025	78.36

Table 4: Recovered Parameters using the Multi–Frame Algorithm (Image 6 $\longrightarrow$ Image 8)

Parameters	*Actual*	38^o	34^o	29^o	25^o	20^o	10^o
Ω_x	-3.0	-3.01	-2.986	-2.787	-2.411	-1.923	2.483
Ω_y	0.0	0.16	0.149	0.165	0.176	0.318	0.015
Ω_z	-5.0	-3.78	-3.729	-3.521	-3.176	-1.983	-0.157
V_x	0.0	0.768	0.722	0.812	1.242	2.724	3.476
V_y	10.0	7.093	7.023	6.835	6.251	4.872	2.489
V_z	20.0	16.518	15.325	14.313	13.826	14.178	11.415
p	4.42	15.124	16.113	16.725	16.937	19.255	17.837
q	78.36	70.502	70.547	68.463	66.165	63.361	61.163

Table 5: Effect of Varying the View Angle on the Recovered Parameters (Image 7 $\longrightarrow$ Image 8)

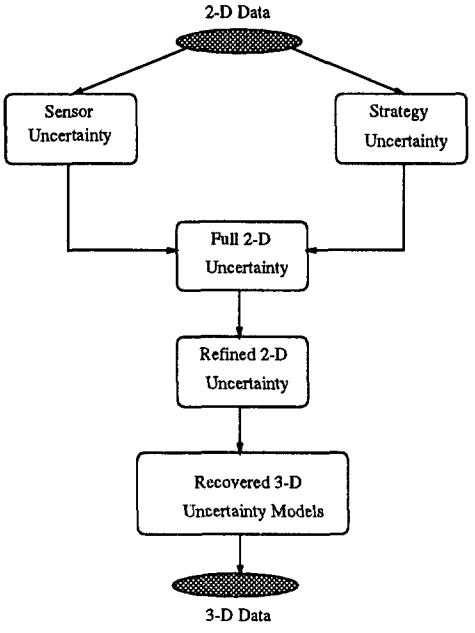


Figure 6: Propagation of Uncertainty

As an example to illustrate the idea, let's assume that we have a linear system of equations as follows :

$$x + 3y = z_1$$

$$2x + y = z_2$$

The solution of this system is very easily obtained as :

$$x = \frac{3}{5}z_2 - \frac{1}{5}z_1$$

$$y = \frac{2}{5}z_1 - \frac{1}{5}z_2$$

That is, a linear combination of the right hand side parameters. If the parameters z_1 and z_2 were random variables of known probability distributions instead of constants, then the problem becomes slightly harder, which is, to find the linear combination of those random variables as another random variable. The obvious way of doing this would be to use convolutions and the formula :

$$P_{X_1+X_2}(y) = \sum_R P_{X_1,X_2}(x, y-x)$$

for the sum of two random variables X_1, X_2 for any real number y and/or the formula for linear combinations over the region R, which is for all x such that $P_{X_1,X_2}(x, y-x) > 0$. Using the moment generating function or the characteristic function seems also to be a very attractive alternative. The moment generating function M of a linear combination of random variables, for example X_1, X_2 can be written as :

$$M_{aX_1+bX_2+c}(t) = e^{ct}\left(M_{X_1}(at)M_{X_2}(bt)\right)$$

for independent random variables X_1, X_2. That is, the problem of solving linear systems on the form $Ax = b$, where b is a vector of random variables, may be reduced to finding closed form solutions for x in terms of the random parameters (using any elimination technique) and then manipulating the results and finding different expectations using moment generating or characteristic functions.

The solutions we suggest to this problem of finding the random variable solution of the 3-D motion and structure parameters utilize the techniques we described in the previous sections. Using either the two-frame iterative technique or the multi-frame algorithm, it should be noticed that the problem reduces to either solving multilinear systems or a single one; but in random variables instead of singular values. In that case, using elimination and characteristic functions for computing the required expectations and distributions is straight forward. As an example, the recovered 3-D translational velocity cumulative density functions (CDF) for an actual world motion of a robot gripper in our experiment equal:

$V_X = 0\ cm$, $V_Y = 0\ cm$ and $V_Z = 13\ cm$
is shown in figure 7. It should be noted that the recovered distributions represents a fairly accurate estimation of the actual 3-D motion.

8 The Experimental Uncertainty Recovery System

The design and the experiments for the proposed uncertainty recovering formulation were performed on the architecture shown in Figure 8. The agent under observation is the Lord experimental gripper and is mounted on a PUMA 560. The robot and the hand are essentially moved by an external operator to perform some actions on a set of objects lying on a table.

The observer sensor is another PUMA 560 on which a camera is mounted. The low level visual feature acquisition is performed on the MaxVideo pipelined video processor at frame rate. In particular, there are two separate paths from the vision sensor. One path is for the computation of the hand 3-D position and orientation and this is done through the MaxVideo. The other path (the inner loop) is done on a SparcStation, in which the image processing modules resides, those modules compute 2-D cues from the scene under observation. Identification of objects, their location with respect to the hand and establishing contact, and correlation procedures are all performed within the inner loop. The 2-D to 3-D conversion and probability computations are performed on another SparcStation. Thus future modifications and enhancements could be coded and executed in a simple and modular fashion. Enhanced Low-level modules for segmentation and 2-D understanding of the image and to accommodate different kinds of objects in the scene could be coded within the inner-loop computer module.

9 Conclusions

The recovery methods described here have a variety of applications. It can be useful in vision-guided applications such as autonomous landing and navigation. It may be a starting point for determining global structure - motion analysis of entire polyhedra, making it suitable for robotics applications in the "moving blocks world". Parallel implementations could be designed for such problems, thus solving for the structure - motion parameters for each surface separately. In fact, solving the linear system at each iteration could also be parallelized. We have also demonstrated that the uncertainty in 2-D sense data can actually be *utilized* for recovering the motion and structure of a scene under observation *robustly.*

10 Acknowledgments

The author wishes to extend his thanks to Professor Kwangyoen Wohn for initiating the work on the Two–Frame and Multi–Frame algorithms and for supplying the preliminary software version for them.

References

[1] N.M. Grzywacz and E.C. Hildreth, *The Incremental Rigidity Scheme for Recovering Structure from Motion: Position vs. Velocity Based Formulations*, MIT A.I. Memo No. 845, October 1985.

[2] S-L. Iu and K. Wohn, "Estimation of 3-D Motion and Structure Based on a Temporally Oriented Approach with the Method of Regression", *IEEE Workshop on Visual Motion*, March 1989, Irvine, CA, 273-281.

[3] H.C. Longuet-Higgins and K.Prazdny, *The interpretation of a moving Retinal Image*, Proc. Royal Society of London B, 208, 385-397, 1980.

[4] M. Subbarao and A.M. Waxman, *On The Uniqueness of Image Flow Solutions for Planar Surfaces in Motion*, CAR–TR–113, Center for Automation Research, University of Maryland, April 1985.

[5] R.Y. Tsai and S.T. Huang, "Estimating three-dimensional motion parameters of a rigid planar patch",*IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-29(6), December 1981.

[6] S. Ullman, *Maximizing Rigidity: The incremental recovery of 3-D structure from rigid and rubbery motion*, AI Memo 721, MIT AI lab. 1983.

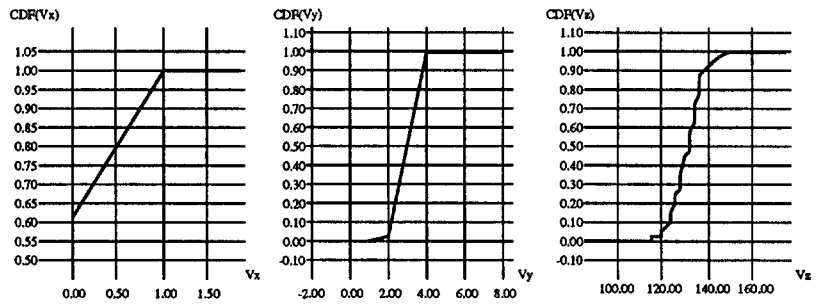


Figure 7: Cumulative Density Functions of the Translational Velocity

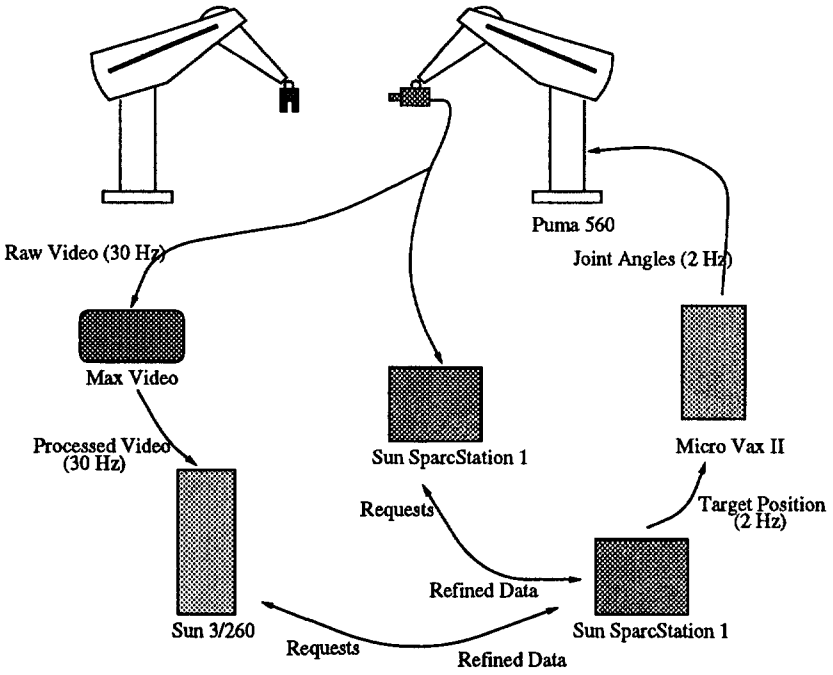


Figure 8: The Architecture of the System

[7] A.M. Waxman and S. Ullman, *Surface Structure and 3-D Motion From Image Flow: A Kinematic Analysis*, CAR–TR–24, Center for Automation Research, University of Maryland, October 1983.

[8] J. Weng, T.S. Huang and N. Ahuja, "3-D Motion Estimation, Understanding and Prediction from Noisy Image Sequences", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(3), May 1987.

Report: PARLE'94 - Parallel Architectures and Languages in Europe

July 2 – 7, 1994, Athens, Greece.

Borut Robič

Parallel processing is now well established within the high performance computing technology and constitutes the main thrust towards the development of new products and solutions which pose demand for large scale computation. From that point of view parallel processing is of strategic importance not only for the informatics industry, but also for a wide area of applications. Porting classical applications to the already existing parallel machines, developing new applications which would be infeasible in the realm of the uniprocessor, and designing new more powerful parallel computers has become the "new world" for the computer scientists, the engineers and the implementors. It is anticipated that the impact of parallelism will be not only on the computer industry, but also on other industrial sectors. The whole economy will be affected by parallelism in the near future.

PARLE is the main scientific event on parallel processing held in Europe. It is an international conference focusing on parallel computer languages, and architectures. Since its origination in 1987 as an initiative coming from ESPRIT I program, it has grown to a major event which has assumed high international reputation and is *the* European forum on parallelism. From 1995 onwards it will continue as EURO-PAR conference, as a result of merging of PARLE and CONPAR/VAPP events.

PARLE'94 was organized in Athens by the Computer Technology Institute at Patras, Greece (C.T.I.) as the sixth in a series of similar events. Authors submitted over 250 papers of which 84 (from 21 countries) were selected and presented at the conference. There were 21 sessions running in two tracks: Interconnection Networks I and II, Compiling Techniques, Special Purpose Systems, Communication Protocols, Algorithms for Multiprocessor Networks, Programming Environments, Scientific Computing, Performance Evaluation, Data Distribution, Cache Systems, Language Issues, Language Implementation, Applications, System Evaluation, Scheduling, Semantics, Load Balancing, Parallel Algorithms, Miscellanea, and Poster Session. Invited talks discussed theoretical and practical issues in structural parallel algorithms (*Uzi Vishkin*), evolution and challenges of multithreaded computer architectures (*Guang Gao*), and parallelism in relational databases (*M. Hoevenaars*). Of the papers, let us mention those dealing with improved probabilistic routing in generalized hypercubes (*A.G.Ferreira, M.D.Grammatikakis*), multisearching problem for hypercubes (*M.J.Attalah, A.Fabri*), hierarchical activation management techniques for fine-grain multithreaded execution (*C.Kim, J.-L. Gaudiot*), array processor architecture for matrix computations (*S.P.S.Lam*), mapping with parallel simulated annealing (*B.Robič, J.Šilc*), performance of interconnection network in multithreaded architectures (*S.S.Nemawarkar, R.Govindarajan, G.R.Gao, V.K.Agarwal*), parallelism of data (*C.V.Papadopoulos*), and some new ideas on the definition of the speedup (*W.Ertel*). Tutorials were given by *Franco P. Preparata* (Models and Fundamental Techniques of Parallel Computation), *Dough Degroot* (Controlling and Limiting Dynamic and Speculative Parallelism), *L.O.Herzberger* (Parallel Computing Architectures), and *Peter Kacsuk* (Parallel Implementations of Logic Programs).

The organization of the PARLE'94 conference was excellent. The proceedings are published in *Lecture Notes in Computer Science*, vol. 817 (Springer-Verlag).

This is a Call For Papers for a special journal issue of INFORMATICA on the topic:

MIND <> COMPUTER

[i.e. Mind NOT EQUAL Computer]

MOTIVATION:
Recent progress in AI (or, as some people would say, the lack of progress) brings to mind papers by Winograd, Dreyfus, etc. that question the possibility of achieving "strong" AI. Was the scientific community correct when rejecting their ideas years ago?

In this special issue we want to re-evaluate the soundness of current AI research positions, especially the heavily disputed strong-AI paradigm, and explore new approaches that aim to achieve true intelligence.

The core of this special issue will be a small number of invited papers, including papers by Winograd, Dreyfus, Michie, McDermott, Agre, Tecuci, etc. Here, we are soliciting additional papers on the topic.

TOPICS:
Papers are invited in all subareas and aspects of the above topic, especially in:

- Current state, positions, and "real" advancements achieved in the last 5 years (as opposed to parametric improvements).
- Trends, perspectives and foundations of natural and artificial intelligence.
- Strong AI versus weak AI versus the reality of most "typical" publications in AI.
- New directions in AI.

FORMAT AND REVIEWING PROCESS:
Papers should not exceed 8,000 words (including figures and tables but excluding references. A full page figure should be counted as 500 words). Ideally 5,000 words are desirable. If accepted, the authors will be invited to transform their manuscripts into the Informatica LaTeX style.

Each paper will be refereed by at least two anonymous referees outside the author's country and by an appropriate subset of the program committee.

TIME TABLE AND CONTACTS:
Papers in 5 hard copies should be received by May 15, 1995 at one of the following addresses (no email/fax submissions):

North & S. America – Jim Geller
New Jersey Institute of Technology
CIS Department
323 Dr. King Blvd:
Newark, NJ 07102, USA
geller@vienna.njit.edu

Asia, Australia – Xindong Wu
Department of Software Development, Monash University,
Melbourne, VIC 3145, Australia
xindong@insect.sd.monash.edu.au

Europe, Africa – Matjaz Gams
Jozef Stefan Institute, Jamova 39,
61000 Ljubljana, Slovenia, Europe
matjaz.gams@ijs.si

E-mail information about the special issue is available from the above 3 contact editors.

The special issue will be published in late 1995. Depending on the number and quality of submissions there is a possibility for the special issue to be reproduced as or extended to a book.

More information about Informatica and the special issue can be accessed through URL: ftp://ftp.arnes.si/magazines/informatica.

THE MINISTRY OF SCIENCE AND TECHNOLOGY OF THE REPUBLIC OF SLOVENIA

Address: Slovenska 50, 61000 Ljubljana, Tel.: +386 61 1311 107, Fax: +386 61 1324 140.
Minister: Prof. Rado Bohinc, Ph.D.
State Secretary for Int. Coop.: Rado Genorio, Ph.D.
State Secretary for Sci. and Tech.: Ciril Baškovič
Secretary General: Franc Hudej, Ph.D.

The Ministry also includes:
The Standards and Metrology Institute of the Republic of Slovenia
Director: Peter Palma
Address: Kotnikova 6, 61000 Ljubljana, Tel.: +386 61 1312 322, Fax: +386 61 314 882.
and
The Industrial Property Protection Office of the Republic of Slovenia
Director: Bojan Pretnar, Ph. D.
Address: Kotnikova 6, 61000 Ljubljana, Tel.: +386 61 1312 322, Fax: +386 61 318 983.

Scientific Research and Development Potential. The statistical data for 1993 showed that there were 180 research and development institutions in Slovenia. Altogether, they employed 10,400 people, of whom 4,900 were researchers and 3,900 expert or technical staff.

In the past ten years, the number of researchers has almost doubled: the number of Ph.D. graduates increased from 1,100 to 1,565, while the number of M.Sc.'s rose from 650 to 1,029. The "Young Researchers" (i.e. postgraduate students) program has greatly helped towards revitalizing research. The average age of researchers has been brought down to 40, with one-fifth of them being younger than 29.

The table below shows the distribution of researchers according to educational level and sectors (in 1993):

Sector	Ph.D.	M.Sc.
Business enterprises	51	196
Government	482	395
Private non-profit organizations	10	12
Higher education organizations	1022	426
Total	1,565	1,029

Financing Research and Development. Statistical estimates indicate that US$ 185 million (1,4% of GDP) was spent on research and development in Slovenia in 1993. More than half of this comes from public expenditure, mainly the state budget. In the last three years, R&D expenditure by business organizations has stagnated, a result of the current economic transition. This transition has led to the financial decline and increased insolvency of firms and companies. These cannot be replaced by the growing number of mainly small businesses. The shortfall was addressed by increased public-sector spending: its share of GDP nearly doubled from the mid-seventies to 0,86% in 1993.

Income of R&D organizations spent on R&D activities in 1993 (in million US$):

Sector	Total	Basic res.	App. res.	Exp. dev.
Business ent.	83,9	4,7	32,6	46,6
Government	58,4	16,1	21,5	20,8
Private non-p.	1,3	0,2	0,6	0,5
Higher edu.	40,9	24,2	8,7	8
Total	184,5	45,2	63,4	75,9

The policy of the Slovene Government is to increase the percentage intended for R&D in its budget.

The Science and Technology Council of the Republic of Slovenia is preparing the draft of a national research program (NRP). The government will harmonize the NRP with its general development policy, and submit it first to the parliamentary Committee for Science, Technology and Development and after that to the parliament. The parliament approves the NRP each year, thus setting the basis for deciding the level of public support for R&D.

The Ministry of Science and Technology is mainly a government institution responsible for controlling expenditure of the R&D budget, in compliance with the NRP and the criteria provided by the Law on Research Activities. The Ministry finances research or co- finances development projects through public bidding, partially finances infrastructure research institutions (national institutes), while it directly finances management and top-level science.

The focal points of R&D policy in Slovenia are:

- maintaining the high level and quality of research activities,
- stimulating collaboration between research and industrial institutions,
- (co)financing and tax assistance for companies engaged in technical development and other applied research projects,
- research training and professional development of leading experts,
- close involvement in international research and development projects,
- establishing and operating facilities for the transfer of technology and experience.

JOŽEF STEFAN INSTITUTE

Jožef Stefan (1835-1893) was one of the most prominent physicists of the 19th century. Born to Slovene parents, he obtained his Ph.D. at Vienna University, where he was later Director of the Physics Institute, Vice-President of the Vienna Academy of Sciences and a member of several scientific institutions in Europe. Stefan explored many areas in hydrodynamics, optics, acoustics, electricity, magnetism and the kinetic theory of gases. Among other things, he originated the law that the total radiation from a black body is proportional to the 4th power of its absolute temperature, known as the Stefan-Boltzmann law.

The Jožef Stefan Institute (JSI) is the leading independent scientific research in Slovenia, covering a broad spectrum of fundamental and applied research in the fields of physics, chemistry and biochemistry, electronics and information science, nuclear science technology, energy research and environmental science.

The Jožef Stefan Institute (JSI) is a research organisation for pure and applied research in the natural sciences and technology. Both are closely interconnected in research departments composed of different task teams. Emphasis in basic research is given to the development and education of young scientists, while applied research and development serve for the transfer of advanced knowledge, contributing to the development of the national economy and society in general.

At present the Institute, with a total of about 700 staff, has 500 researchers, about 250 of whom are postgraduates, over 200 of whom have doctorates (Ph.D.), and around 150 of whom have permanent professorships or temporary teaching assignments at the Universities.

In view of its activities and status, the JSI plays the role of a national institute, complementing the role of the universities and bridging the gap between basic science and applications.

Research at the JSI includes the following major fields: physics; chemistry; electronics, informatics and computer sciences; biochemistry; ecology; reactor technology; applied mathematics. Most of the activities are more or less closely connected to information sciences, in particular computer sciences, artificial intelligence, language and speech technologies, computer-aided design, computer architectures, biocybernetics and robotics, computer automation and control, professional electronics, digital communications and networks, and applied mathematics.

The Institute is located in Ljubljana, the capital of the independent state of Slovenia (or S♡nia). The capital today is considered a crossroad between East, West and Mediterranean Europe, offering excellent productive capabilities and solid business opportunities, with strong international connections. Ljubljana is connected to important centers such as Prague, Budapest, Vienna, Zagreb, Milan, Rome, Monaco, Nice, Bern and Munich, all within a radius of 600 km.

In the last year on the site of the Jožef Stefan Institute, the Technology park "Ljubljana" has been proposed as part of the national strategy for technological development to foster synergies between research and industry, to promote joint ventures between university bodies, research institutes and innovative industry, to act as an incubator for high-tech initiatives and to accelerate the development cycle of innovative products.

At the present time, part of the Institute is being reorganized into several high-tech units supported by and connected within the Technology park at the Jožef Stefan Institute, established as the beginning of a regional Technology park "Ljubljana". The project is being developed at a particularly historical moment, characterized by the process of state reorganisation, privatisation and private initiative. The national Technology Park will take the form of a shareholding company and will host an independent venture-capital institution.

The promoters and operational entities of the project are the Republic of Slovenia, Ministry of Science and Technology and the Jožef Stefan Institute. The framework of the operation also includes the University of Ljubljana, the National Institute of Chemistry, the Institute for Electronics and Vacuum Technology and the Institute for Materials and Construction Research among others. In addition, the project is supported by the Ministry of Economic Relations and Development, the National Chamber of Economy and the City of Ljubljana.

Jožef Stefan Institute
Jamova 39, 61000 Ljubljana, Slovenia
Tel.:+386 61 1259 199, Fax.:+386 61 219 385
Tlx.:31 296 JOSTIN SI
WWW: http://www.ijs.si
E-mail: matjaz.gams@ijs.si
Contact person for the Park: Iztok Lesjak, M.Sc.
Public relations: Natalija Polenec

CONTENTS OF *Informatica*, Volume 18 (1994) pp. 1–510

Articles

Profiles

News and Conferences

Professional Journals

Professional Societies

Železnikar, A.P.: *World Organisation of Systems and Cybernetics,* Informatica **18** (1994) 245.

Jožef Stefan Insitute, Informatica **18** (1994) 105, 254, 373, and 504.

The Ministry of Science and Technology of the Republic of Slovenia, Informatica **18** (1994) 104, 253, 372, and 503.

Information for Contributors, Informatica **18** (1994) 106.

REVIEW REPORT

Basic Instructions

Informatica publishes scientific papers accepted by at least two referees outside the author's country. Each author should submit three copies of the manuscript with good copies of the figures and photographs to one of the editors from the Editorial Board or to the Contact Person. Editing and refereeing are distributed. Each editor can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. The names of the referees should not be revealed to the authors under any circumstances. The names of referees will appear in the Refereeing Board. Each paper bears the name of the editor who appointed the referees.

It is highly recommended that each referee writes **as many remarks as possible directly on the manuscript**, ranging from typing errors to global philosophical disagreements. The chosen editor will send the author copies with remarks, and if accepted also to the Contact Person with the accompanying completed Review Reports. The Executive Board will inform the author that the paper is accepted, meaning that it will be published in less than one year after receiving original figures on separate sheets and the text on an IBM PC DOS floppy disk or through e-mail – both in ASCII and the Informatica LaTeX format. Style and examples of papers can be obtained by e-mail from the Contact Person or from FTP or WWW (see the last page of Informatica).

Date Sent:

Date to be Returned:

Name and Country of Referee:

Signature of Referee:

Name of Editor:

Title:

Authors:

Additional Remarks:

All boxes should be filled with numbers 1-10 with 10 as the highest rated.

The final mark (recommendation) consists of two orthogonal assessments: scientific quality and readability. The readability mark is based on the estimated perception of average reader with faculty education in computer science and informatics. It consists of four subfields, representing if the article is interesting for large audience (interesting), if its scope and approach is enough general (generality), and presentation and language. Therefore, very specific articles with high scientific quality should have approximately similar recommendation as general articles about scientific and educational viewpoints related to computer science and informatics.

☐ SCIENTIFIC QUALITY

☐ Originality

☐ Significance

☐ Relevance

☐ Soundness

☐ Presentation

☐ READABILITY

☐ Interesting

☐ Generality

☐ Presentation

☐ Language

☐ FINAL RECOMMENDATION

☐ Highly recommended

☐ Accept without changes

☐ Accept with minor changes

☐ Accept with major changes

☐ Author should prepare a major revision

☐ Reject

INFORMATICA

AN INTERNATIONAL JOURNAL OF COMPUTING AND INFORMATICS

INVITATION, COOPERATION

Submissions and Refereeing

Please submit three copies of the manuscript with good copies of the figures and photographs to one of the editors from the Editorial Board or to the Contact Person. At least two referees outside the author's country will examine it, and they are invited to make as many remarks as possible directly on the manuscript, from typing errors to global philosophical disagreements. The chosen editor will send the author copies with remarks. If the paper is accepted, the editor will also send copies to the Contact Person. The Executive Board will inform the author that the paper has been accepted, in which case it will be published within one year of receipt of the original figures on separate sheets and the text on an IBM PC DOS floppy disk or by e-mail – both in ASCII and the Informatica LaTeX format. Style and examples of papers can be obtained by e-mail from the Contact Person or from FTP or WWW (see the last page of Informatica).

Opinions, news, calls for conferences, calls for papers, etc. should be sent directly to the Contact Person.

QUESTIONNAIRE

☐ Send Informatica free of charge

☐ Yes, we subscribe

Please, complete the order form and send it to Dr. Rudi Murn, Informatica, Institut Jožef Stefan, Jamova 39, 61111 Ljubljana, Slovenia.

Since 1977, Informatica has been a major Slovenian scientific journal of computing and informatics, including telecommunications, automation and other related areas. In its 16th year (two years ago) it became truly international, although it still remains connected to Central Europe. The basic aim of Informatica is to impose intellectual values (science, engineering) in a distributed organisation.

Informatica is a journal primarily covering the European computer science and informatics community - scientific and educational as well as technical, commercial and industrial. Its basic aim is to enhance communications between different European structures on the basis of equal rights and international refereeing. It publishes scientific papers accepted by at least two referees outside the author's country. In addition, it contains information about conferences, opinions, critical examinations of existing publications and news. Finally, major practical achievements and innovations in the computer and information industry are presented through commercial publications as well as through independent evaluations.

Editing and refereeing are distributed. Each editor can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. If new referees are appointed, their names will appear in the Refereeing Board.

Informatica is free of charge for major scientific, educational and governmental institutions. Others should subscribe (see the last page of Informatica).

ORDER FORM – INFORMATICA

Name: .....

Title and Profession (optional): .....

.....

Home Address and Telephone (optional): .....

.....

Office Address and Telephone (optional): .....

.....

E-mail Address (optional): .....

Signature and Date: .....

EDITORIAL BOARDS, PUBLISHING COUNCIL

Informatica is a journal primarily covering the European computer science and informatics community; scientific and educational as well as technical, commercial and industrial. Its basic aim is to enhance communications between different European structures on the basis of equal rights and international refereeing. It publishes scientific papers accepted by at least two referees outside the author's country. In addition, it contains information about conferences, opinions, critical examinations of existing publications and news. Finally, major practical achievements and innovations in the computer and information industry are presented through commercial publications as well as through independent evaluations.

Editing and refereeing are distributed. Each editor from the Editorial Board can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. If new referees are appointed, their names will appear in the Refereeing Board. Each paper bears the name of the editor who appointed the referees. Each editor can propose new members for the Editorial Board or Board of Referees. Editors and referees inactive for a longer period can be automatically replaced. Changes in the Editorial Board and Board of Referees are confirmed by the Executive Editors.

The coordination necessary is made through the Executive Editors who examine the reviews, sort the accepted articles and maintain appropriate international distribution. The Executive Board is appointed by the Society Informatika. Informatica is partially supported by the Slovenian Ministry of Science and Technology.

Each author is guaranteed to receive the reviews of his article. When accepted, publication in Informatica is guaranteed in less than one year after the Executive Editors receive the corrected version of the article.

Volume 18 Number 4 December 1994 ISSN 0350-5596

Informatica

An International Journal of Computing and Informatics

Contents: