GENETIC PROGRAMMING
Poster Papers

Riccardo Poli, chair

Comparison of Evolving Against Peers and Fixed Opponents Using Corewars

Jason Cooper
Department of Computer Science
Loughborough University
LEICS, LE11 3TU
j.l.cooper@lboro.ac.uk

Chris Hinde
Department of Computer Science
Loughborough University
LEICS, LE11 3TU
c.j.hinde@lboro.ac.uk

Abstract

Two methods of evolving Corewars programs are compared, one against a fixed set of opponents, another against the other programs in the generation. The fixed opponent system improves faster initially but is limited overall. The second is slower to evolve but achieves a better final result.

EVOLVING COREWARS WARRIORS

Corewars[1] is a game where two programs written in a language called redcode, try to destroy each other. The programs fight against each other in a simulator. A program wins when all of its opponent's processes have terminated with invalid instructions.

A group of warriors evolved against fixed opponents (Group F) was compared with a similar group evolving against their peers (Group P). An unseen control set of 10 fixed opponents (Group C), was used as a benchmark to compare the other two groups giving a common fitness indicator for both sets. The warriors in the control group had competed in previous international Corewars tournaments in 1989 and 1990.

The values shown on the graph in Figure 1 are the average fitness level of Group P and Group F, over 300 generations, when tested against Group C. The initial performance of Group F can be explained by the more stable environment they are in. Later on though, Group F reach a stage where they are getting reasonable results most of the time against their fixed opponents. Individuals in Group P do not stay ahead of one another for long as the best strategies propagate through the rest of the population over the next few generations and so any successful individual must find a better strategy to enable them to win. The strategies evolved by both groups are transferable as neither group has any knowledge of the control group.

Figure 1: Fitness levels of Group P and Group F

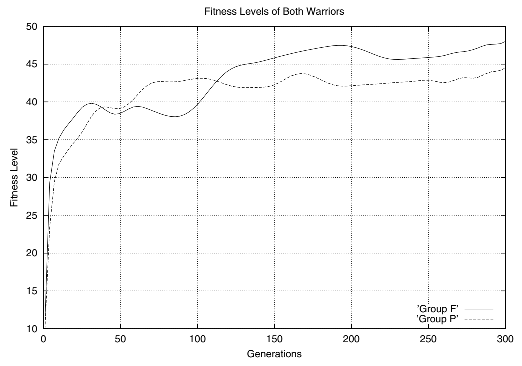

Table 1: League table of Group C and the best individual in Group P and Group F at generation 350

Position	Warrior	Fitness
1..4	Control Opponent 6,4,9,8	91 .. 122
5	**Best Individual of Group P after 350 Generations**	71
6	Control Opponent 10	62
7	**Best Individual of Group F after 350 Generations**	61
8..12	Control Opponent 3,5,2,1,7	41 .. 57

Acknowledgements

The authors of this paper would like to acknowledge the support of Nortel Networks.

References

[1] Dewdney A.K., 1990, The Magic Machine: A handbook of Computer Sorcery. ISBN 0-7167-2125-2.

Open BEAGLE: A New C++ Evolutionary Computation Framework

Christian Gagné and **Marc Parizeau**
Laboratoire de Vision et Systèmes Numériques (LVSN)
Université Laval, Québec (QC), Canada, G1K 7P4.
E-mail: {cgagne,parizeau}@gel.ulaval.ca

Abstract

This poster introduces a new C++ Evolutionary Computation (EC) framework named *Open BEAGLE*. This framework is freely available on the projet's Web page at `http://www.gel.ulaval.ca/~beagle`.

Open BEAGLE[1] is a C++ framework for doing almost any kind of EC. Its architecture follows the principles of Object Oriented (OO) programming, where some abstractions are represented by loosely coupled objects and where it is common and easy to reuse code. Open BEAGLE has a three level architecture as illustrated by Figure 1. The OO foundations are the basis of this architecture, as an object oriented extension of C++, inspired by *design patterns* (Gamma et al., 1994; Lenaerts and Manderick, 1998). It offers basic functionalities like smart pointers and garbage collection, object allocators, standard containers, and XML I/O streams. The generic EC framework implements basic mechanisms and structures for designing versatile specialized Evolutionary Algorithms (EA). It is summarized in Figure 2. It comprises three main components: a vivarium, an evolution system, and an evolver. The vivarium is a container for demes of generic individuals. The individuals themselves are specified by an abstract genotype. This genotype can be instantiated to any relevant structure (in Figure 2, it is shown as a bit string, but this is just an example). Individuals and demes can also be specialized if needed.

In the evolution system, the *context* contains the state of the genetic engine, such as the current deme and generation number. This concept is similar to the execution context of a computer. The *register* is a central repository for all evolution parameters. The evolving process itself is governed by an *evolver* that defines sequences of operations, contained in operator sets,

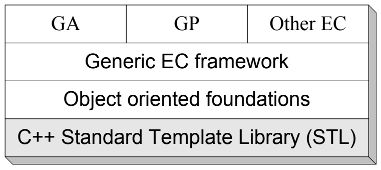
GA
GP
Other EC
Generic EC framework
Object oriented foundations
C++ Standard Template Library (STL)

Figure 1: Open BEAGLE Framework Architecture.

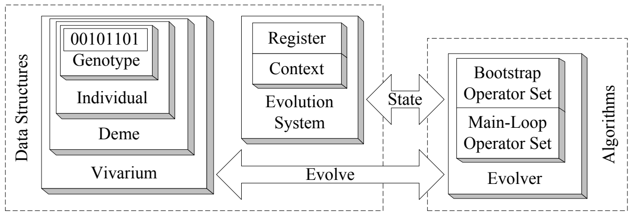

Figure 2: Generic EC Framework Architecture

that are iteratively applied to demes. The evolver applies the *bootstrap operator set* to initialize the first generation, and the *main-loop operator set* to the subsequent generations. For common EA, standard operators have been defined, such as common selection schemes, crossovers, mutations, statistics calculation, and evolution checkpoint backup.

The specialized frameworks are at the top level of the architecture. Currently, only classical genetic algorithms and genetic programming frameworks have been implemented. But we are looking forward to external contributions by interested researchers for other specialized EA. Extensive documentation is available on the Open BEAGLE's Web page.

[1] The recursive acronym BEAGLE means *the Beagle Engine is an Advanced Genetic Learning Environment.*

References

Gamma, E., Helm, R., Johnson, R., and Vlissides, J.: 1994, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley

Lenaerts, T. and Manderick, B.: 1998, Building a genetic programming framework: The added-value of design patterns, in *Proceedings of EuroGP '98*, Vol. 1391 of *LNCS*, pp 196–208, Springer-Verlag

How Statistics Can Help in Limiting the Number of Fitness Cases in Genetic Programming

Mario Giacobini, Marco Tomassini, Leonardo Vanneschi
Institut d'Informatique, University of Lausanne, 1015 Lausanne, Switzerland
{Mario.Giacobini,Marco.Tomassini,Leonardo.Vanneschi}@iis.unil.ch

For most real world applications of GP it is well known that fitness evaluation is the most time consuming operation. It would thus be interesting to establish criteria that can help in limiting the time spent in this phase as much as possible without compromising results in terms of quality. One is thus confronted with two problems: how to select a sufficient number of fitness cases and how to choose those fitness cases in such a way that they are effective in driving the learning process towards a solution. Here we approach the former problem from a standard statistical and information-theoretical viewpoint.

Let us consider a GP problem where the target function is defined on N fitness cases. It can be shown that the mean distance of all the individuals of a population from the target function is normally distributed ($\bar{x} \sim N(\mu, \sigma)$). A standard result for the confidence interval gives [2]:

$$P\left(\bar{x} - t_{\alpha/2}\left(\frac{\sigma}{\sqrt{n}}\right) < \mu < \bar{x} + t_{\alpha/2}\left(\frac{\sigma}{\sqrt{n}}\right)\right) \geq 1 - \alpha,$$

where $1-\alpha$ is the confidence with which we can expect the mean μ to be contained in the given interval. The $t_{\alpha/2}$ is the Student cumulative distribution such that the mean deviates from its true value in the interval $(-t_{\alpha/2}, t_{\alpha/2})$. The standard deviation σ is unknown but can be estimated by the sample variance S. If we set $K = 2t_{\alpha/2}(\sigma/\sqrt{n})$, the length of the confidence interval, we get a function relating the number of fitness cases n that must be used in order for the mean fitness to be estimated to be in the confidence interval K with a given probability $1 - \alpha$.

The target function $g : \{x_1, ..., x_N\} \to \{y_1, ..., y_M\}$ can be seen, from another viewpoint as a random variable, and it is thus possible to calculate its entropy:

$$H(g(x)) = -\frac{1}{ln(N)} \sum_{j=1}^{M} p_j \ln(p_j),$$

where $p_j = P(g(x) = y_j)$ for $j \in \{1, ..., M\}$. Such a measure indicates the quantity of information needed to determine the function itself, i.e., the minimal number of fitness cases needed for a reliable reconstruction of the target function g.

To test the validity of our assumptions we have studied two simple problems: a seven variables boolean function, and a step function. The aim is to show the statistical behavior of the GP evolutions when the number of fitness cases is decreased. For such a purpose standard GP has been run 50 times for each percentage of fitness cases, randomly chosen with uniform probability. For both target functions we observe a similar statistical behavior. When the number of fitness cases is such that the level of confidence is 0.99 we observe a normal convergence behavior, while with a number of fitness cases lower than such a value we get oscillating curves and the length of the confidence interval drastically increases. Such a result is consistent with the entropy which is found to be close to that value of n. For the boolean function the minimal n is about 18 with respect to 128 fitness cases, while for the step function we get 27 instead of the full 100 cases.

Our results are of a statistical nature and thus they do not depend on the particular problem. Some previous works have tackled the problem of limiting the number of fitness cases heuristically (e.g. [1]). Knowing that the number of fitness cases can be significantly reduced for statistical reasons can be useful for selecting a reduced but sufficient number of significant fitness cases.

References

[1] C. Gathercole and P. Ross. Tackling the boolean even N parity problem with genetic programming and limited-error fitness. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 119–127, San Francisco, CA, USA, 1997. Morgan Kaufmann.

[2] S. M. Ross. *Introduction to Probability and Statistics for Engineers and scientists.* Academic Press, New York, 2000.

A New Model to Realize Variable Size Genetic Network Programming

Hironobu Katagiri, Kotaro Hirasawa, Jinglu Hu and Junichi Murata
Department of Electrical and Electronic Systems Engineering, Kyushu University,
6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan

Genetic Network Programming (GNP) [1] is an extension of Genetic Programming motivated by the strong expression ability of graph. A program in GNP is an arbitrary directed graph, composed of nodes connected to each other by directed arcs. Figure 1 shows the basic scheme of GNP system. Previously, the program size of GNP was fixed. In the paper, a new method is proposed, where the program size is adaptively changed depending on the frequency of use of nodes. Generally, large programs have high expression ability, while their evolutions are disturbed by their enormous search spaces, also they occupy many memory resources and consume much calculation time. To control and to decide a proper program size are important and difficult problems in Evolutionary Computation. We introduce two additional operators, *add operator* and *delete operator*, that can change the number of each kind of node functions in a GNP program based on the degree of contribution of a node function to the fitness value.

A GNP program is composed of *judgement nodes* (Js), *processing nodes* (Ps), and a specific *start node*. A judgement node performs a kind of judgement function, then it transfers control to one of nodes connected by its arcs according to the judgement result. That is, the judgement nodes are conditional branch decision functional nodes. A processing node performs a kind of action function, then it transfers control to a node connected by its arc. A program run starts from the start node. The continuous control flow through a GNP program can express an intelligent and complicated program, because a graph can include the temporal information and sequential information implicitly.

The basic concepts of the proposed two operators are the following simple and valid rules. A frequently used node function through a program run would be an important node function, then the node function should be added; on the contrary, a scarcely used node could be unimportant node, then the node should be deleted. That is, the proposed method varies program sizes not by applying genetic operators randomly, but by taking account of the interaction between the current program and the environment properly.

Simulation results show that the proposed method is clearly of great advantage to evolve GNP programs. It seems that the proposed method saves the time to seek the proper initial program size and reduces the anxiety of the program bloat. Moreover, these additional operators can automatically distinguish important node functions from unimportant node functions for the target environment.

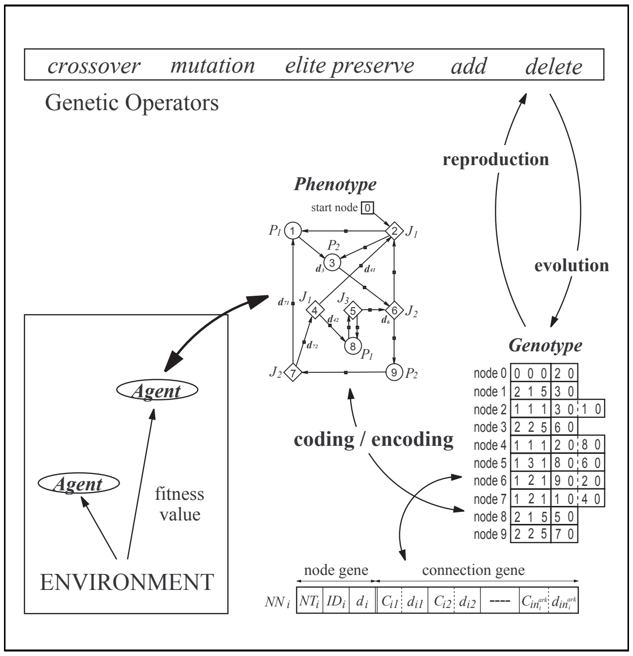

Figure 1: The Basic Scheme of GNP System

References

[1] Katagiri, H., Hirasawa, K., Hu, J., and Murata, J. (2001). Network structure oriented evolutionary model-genetic network programming-and its comparison with genetic programming. In Goodman, E. D., editor, *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 219–226, San Francisco, California, USA.

Controlling the Genetic Programming Search

Emin Erkan Korkmaz
Department of Computer Engineering
Middle East Technical University
Ankara-Turkey
korkmaz@ceng.metu.edu.tr

Göktürk Üçoluk
Department of Computer Engineering
Middle East Technical University
Ankara-Turkey
ucoluk@ceng.metu.edu.tr

Traditional GP randomly combines subtrees by applying crossover. In this study a new approach is presented for guiding the recombination process. Our method is based on extracting the global information of the promising solutions that appear during the genetic search. The aim is to use this information to control the crossover operation afterwards. [1] proposes a method based on calculating the performance values for subtrees of a GP tree during evolution and then applying recombination so that the subtrees with high performance are not disturbed. The aim is to control recombination by determining the building blocks. However for the deceptive class of problems focusing on the building blocks is questionable. The interaction between the partial solutions is high for these problems. Hence it is difficult to determine isolated building blocks.

The frequency information of the elements and their distribution in the tree have been used to determine the global information of a GP tree. The information is represented as a vector. In order to transform a GP tree to a vector, the elements are mapped to base vectors first and a bottom up construction is used to obtain a single vector for the whole tree. A leaf node is only mapped to its base vector while the vector for an internal node is obtained by adding the vectors of its children plus the base vector corresponding to it. Note that the dimension of the vector is the total number of elements used for the problem at hand. However this formalization would enable us to hold only the frequency information. To represent the distribution information too, the depth knowledge is used. This new information is represented as a fractional value to make a distinction with the frequency information. For example if $X(X_1, X_2)$ is a subpart of a three, then the vector corresponding to X is determined as:

$$V_X = V_{X_1} + V_{X_2} + V_{X_{base}} + V_{X_{base}} * 0.01 * depth(X)$$

A *Control Module* is designed to process this global information. The genetic search is started and for each chromosome the corresponding vector is formed. The control module collects the vectors and fitnesses for a certain period of generations, which we call the learning period. Then *"C4.5, Decision Tree Generator"* is used to generate an abstraction over the data collected. Then for each crossover, the genetic engine sends to the control module three different alternative crossover points. The control module predicts if the alternative offsprings will be in the positive or the negative class. Certainly the best alternative is chosen and the learning process is repeated at the end of each learning period. CFG induction and the N-Parity Problem have been selected as the testbeds. Both of them can be considered as highly deceptive. Figure 1 presents the progress obtained for the N-Parity problem. The results denote the average best fitness change for basic GP and for our method. Eight different runs are used to calculate the averages. An improvement has been achieved for the CFG induction problem too.

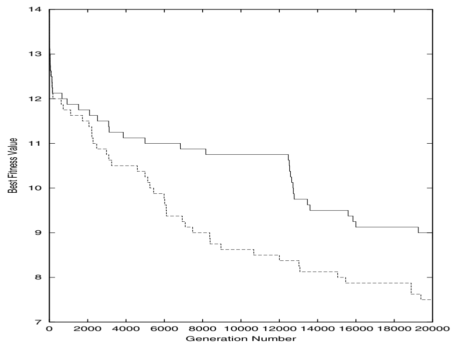

Figure 1: The dashed lines denote the performance of controlled search. Learning period is 500.

References

[1] Hitoshi Iba and Hugo de Garis, *Extending Genetic Programming with Recombinative Guidance*, In P. Angeline and K. E. Kinnear, Jr., editors, Advances in Genetic Programming 2, chapter 4. MIT Press, Cambridge, MA, USA, 1996.

MB GP IN MODELLING AND PREDICTION

Carlos Oliver-Morales
DISCA-IIMAS-UNAM
Circuito Escolar, Cd. Universitaria
O4510 Mexico City, MEXICO

Katya Rodríguez Vázquez
DISCA-IIMAS-UNAM
Circuito Escolar, Cd. Universitaria
O4510 Mexico City, MEXICO

Abstract. *The paper describes a hybrid approach for dynamic system modelling. This proposal is mainly based on a Least Squares algorithm and a Multi-Branch Genetic Programming (MB-GP) encoding. Having multiple branches representing an individual allows us to get simpler mathematical expressions, and therefore, reduces the computational evaluation time.*

1 INTRODUCTION

The encoding proposed in this work is illustrated in Figure 1. It is composed of an *n* number of branches, *n+1* coefficients (one for each branch and the constant term) and the addition function. The branches are mathematical expressions representing function terms and encoding as traditional GP structures; however, the maximum depth of theses branches are much more lower than the one of an approach using traditional GP encoding. Coefficients of each branch are estimated by means of a Least Squares algorithm. The addition function has the aim of adding the product of each branch with its associated coefficient in order to construct the model.
Crossover operator was specifically defined for the MB-GP encoding. Crossing over two parent individuals consists of selecting randomly a branch in each parent and swapping selected branches. Mutation consists of randomly selecting a branch, eliminating the selected branch, and finally, substituting it for a new branch created randomly from primitive functions.

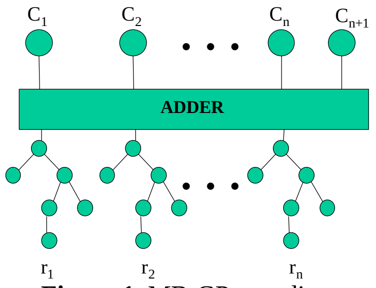

Figure 1. MB-GP encoding.

2 PRELIMINARY RESULTS

The data used in this work (local behaviour of temperature in a large period of time) was measured near Mexico City, at Texcoco Lake. Temperature (T), relative humidity (H), solar radiation (R) and wind speed (V) and direction (D) were recorded. The time interval was 15 minutes.

Based on gathered data, 15% of information corresponding to 324 observations was used for modelling. Experiments were carried out depending on each encoding (MB-GP and Koza-style GP). Cost function was predictive error-based metric. For each experiment, 20 runs were evaluated in order to provide relevant statistical information. The model, which exhibited the best performance from 20 runs, is shown and used to predict future observations (testing data). Based on Koza-style GP (Koza, 1992), the following expression emerged:

(+ (divd T_{T-1} (- (+ (- (.* V_{T-2} R_{T-2}) (- H_{T-1} R_{T-3})) (divd (divd (exp V_{T-2}) (exp D_{T-3})) (.* (- TT-1 D_{T-2}) (divd 0.54707 D_{T-3})))) (cos (sin (sin (sin V_{T-2})))))) (+ T_{T-1} (divd (exp (divd -0.77074 D_{T-1})) (- (+ (exp (exp V_{T-3})) (cos R_{T-2})) (cos (cos (.* D_{T-2} T_{T-2})))))))

In this case, translating previous expression into a mathematical function turns into a difficult task due to complexity in structure. It is also clear that this expression corresponds to a complex function.
In the case of MB-GP, the following expression was generated,

(model T_{T-1} R_{T-1} (.* T_{T-2} R_{T-1}) R_{T-3} (.* R_{T-3} T_{T-2}) 0.95716 0.021437 -0.00087839 -0.020309 0.00090912 0.02055)

Equivalent to,

$$\begin{aligned} T(t) = {} & 0.0255 + 0.9572\, T(t-1) + 0.0214\, R(t-1) - 0.0203\, R(t-3) \\ & - 0.0008\, T(t-2)R(t-1) + 0.0009\, T(t-2)R(t-3) \end{aligned}$$

3 CONCLUSIONS AND FUTURE WORK

An alternative representation in GP for dynamic system modelling and prediction was presented. This MB-GP approach has used small values of GP parameters but these preliminary results showed this encoding could produce simple functions and reduce the search space without penalising complex solutions. MB-GP showed also to be consistent in all experiments.

ACKNOWLEDGEMENTS

Authors gratefully acknowledge the financial support of CONACyT under the project J34900-A. They also thank Dr. Rojano who provided the data.

REFERENCES

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

Self-Improvement for the ADATE Automatic Programming System

Roland Olsson
Computer Science Dept.
Østfold College
1750 HALDEN, Norway
Roland.Olsson@hiof.no
+47 69215369
http://www-ia.hiof.no/~rolando

Brock Wilcox
Northern Arizona University
NAU Box 8087
Flagstaff, AZ 86011, USA
rbw3@cet.nau.edu
928 - 523 - 4903

Automatic Design of Algorithms Through Evolution (ADATE) [2] is a system for automatic programming based on the neutral theory of evolution [1] . This theory states that the majority of molecular changes in evolution are due to neutral or almost neutral mutations. A consequence is that most of the variability and polymorphism within a species comes from mutation-driven drift of alleles that are selectively neutral or nearly neutral. In ADATE, as well as in natural evolution, neutral walks in genotype space are essential for avoiding combinatorial explosions due to complex mutations.

The compound transformations in ADATE were designed according to this model of transition from one species to the next. In ADATE, the first part of a compound program transformation is the neutral walk and the typically small second part is the brutal mutation.

We have taken aim at using ADATE to improve itself by evolving better transition (neutral transforms and mutations) operators. There are many different paths to self-improvement of these transition operators. We explore two such methods which we've called SIG-reshaping and SIG-rewriting.

SIG-rewriting is the automatic synthesis of semantics-preserving rewriting rules that are employed for neutral random walks. Such rules increase the number of connections in a neutral network and thereby also the number of genotypes reachable through a neutral walk without fitness computation.

Since ADATE is optimized to exploit neutrality, it is more difficult to improve ADATE using SIG-rewriting. An advantage of automatically synthesized rewrite rules, however, is that they can be optimized to the specific problem in a way that could not be anticipated when we designed ADATE. Employing several synthesized rewriting rules in ADATE showed no significant performance increase.

The primary goal of SIG-reshaping is to alter the distribution of synthesized (mutated) expressions so that they cover as many equivalence classes as possible. For example synthesizing and using both `E` and `not(not(E))` or both `or`(E_1,E_2) and `or`(E_2,E_1) for arbitrary boolean expressions `E`, E_1 and E_2 may be a waste of time. SIG-reshaping synthesizes an acceptance predicate that determines if a given synthesized expression is used.

The fitness function used in our experiments requires that a synthesized acceptance predicate accepts at least one minimum size expression in each equivalence class and rejects as many other expressions as possible. Given this fitness function ADATE generated an acceptance predicate that rejects 999 out of the 1055 expressions of size five or less consisting of `not`, `and`, `or`, `false`, `true` and three variables `X1`, `X2`, `X3` – while still accepting at least one minimum size member of each class.

SIG-reshaping has demonstrated some self-improvement. After generating an acceptance predicate for boolean expressions, limiting the use of equivalent expressions, the predicate was tested on four, five, and six bit xor problems. Comparing the results with and without the predicate indicate that self-improvement has occurred.

Our web site contains an ADATE specification file for SIG-reshaping as well as the source code of ADATE itself, which should make it easy to reproduce our results.

References

[1] J.L. King and T. H. Jukes (1969). Non-Darwinian evolution. *Science* **164** : 788–798.

[2] J. R. Olsson (1995). Inductive functional programming using incremental program transformation. *Artificial Intelligence.* Vol. 74, No. 1, 55–83.

Evolving Readable Perl

Mark S. Withall
Department of Computer Science
Loughborough University
Leics. LE11 3TU, UK
m.s.withall2@lboro.ac.uk

Chris J. Hinde
Department of Computer Science
Loughborough University
Leics. LE11 3TU, UK
c.j.hinde@lboro.ac.uk

Roger G. Stone
Department of Computer Science
Loughborough University
Leics. LE11 3TU, UK
r.g.stone@lboro.ac.uk

1 INTRODUCTION

A program is informally deemed readable, for the purpose of this experiment, if it is easy for a person to follow the steps that the program takes to solve the problem. In this experiment, readability is achieved by constraining the available syntax for generating solutions.

The Genetic Programming (GP) system created uses the target language Perl because it is an interpreted, untyped, robust procedural language which has good error handling and recovery.

2 GENETIC PROGRAM

The *genotype* and *phenotype* have been separated to make genetic manipulation simpler. Each program is represented as a fixed-length integer array and then mapped onto Backus-Naur Form (BNF). The program statements used are shown in Figure 1a. The BNF is designed to minimise the size of the genome that describes a program. The mapping, between the genotype and phenotype, is similar to *Grammatical Evolution*[2].

The GP was tested using the symbolic regression problem $X^4 + X^3 + X^2 + X$[1]. A population size of 500 and mutation rate of 1 gene in 5000 were used for the test problem. The population was initialised randomly and each test run was of 100 generations. The fitness values for the programs were given as the absolute difference between the target value and the actual value.

3 RESULTS AND CONCLUSIONS

All results were of the correct order (X^4) and 5 out of the 8 test runs produced entirely correct solutions. An example of an optimal program is given in Figure 1b.

The results of the experiment were encouraging. As a comparison the solution evolved by Koza[1] is given in Figure 1c, which is only really understandable by LISP users.

STMT	FORMAT
Assign	$X = Y$
Add	$X = Y + Z$
Sub	$X = Y - Z$
Mul	$X = Y \times Z$
If	if(X cmp Y){
For	for X(0..Y){
End	}

(a)

```
# Header
$x = $ARGV[0];
$res = 0;

# Evolved Code
$res = $x * $x;
$x = $x + $res;
$res = $x * $res;
$res = $res + $x;

# Footer
print "$res";
```

(b)

```
(+X(*(+X(*(*(+X(-(COS(-XX))(-XX)))X)X))X))
```

(c)

Figure 1: (a) List of program statements used. (b) Example of code produced to solve the problem. (c) LISP result from Koza[1].

Acknowledgements

Thanks to everybody and all their friends.

References

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

[2] Ryan C. O'Neill M. & Collins J.J. (1998). Grammatical Evolution: Evolving Programs for an Arbitrary Language. Lecture Notes in Computer Science 1391. First European Workshop on Genetic Programming 1998.