Knowledge Based Evolutionary Programming for Inductive Learning in First-Order Logic

Federico Divina and Elena Marchiori
Department of Mathematics and Computer Science
Vrije Universiteit, Amsterdam, The Netherlands
email: {divina,elena}@cs.vu.nl

Learning from examples in First-Order Logic (FOL), also known as Inductive Logic Programming (ILP) , constitutes a central topic in Machine Learning, with relevant applications to problems in complex domains like natural language and molecular computational biology [Muggleton, 1999].

This paper introduces a knowledge based evolutionary program for inductive concept learning in (a fragment of) FOL, called KBGL (Knowledge Based Genetic Learner). KBGL adopts a Michigan's approach, which means that every individual represents one rule, and individuals co-operate and compete in the evolutionary process. The algorithm uses (a subset of) the background knowledge for constructing the initial population of clauses, and uses novel generalization/specialization operators, based on ILP concepts, in the mutation process. At the end of the evolutionary process, the best subset of clauses (a logic program) is extracted from the final population by means of a fast heuristic procedure. KBGL employs two restriction biases which are controlled by the user: a language bias for reducing the size of the search space, and a stochastic bias on the background knowledge for reducing the computational resources used during the search. At each iteration the probabilistic tournament selection mechanism is used for selecting a number of individuals. Every selected individual undergoes the mutation process, which consists of the repeated application of one, among four, greedy operator until the fitness of the individual increases (the fitness is intended to be minimized), or a maximum number of iterations is reached. If the fitness has increased after the application of a mutation, then the last mutation applied is retracted. As stated before, KBGL uses four mutation operators, namely two operators are used for generalizing a clause , while the other two are used for the specialization of the clause . Every time an individual has to be mutated, the choice of the particular operator to be applied, is made taking into account the features of the individual, in particular the number of positive and negative examples covered by the individual. The chosen operator then will try to apply the best mutation in order to reduce the fitness of the individual.

The method is tested on two ILP case studies: learning illegal white-to-move positions in the chess endgame White King and Rook versus Black King, and learning the mutagenic activity of nitroaromatic compounds. On these problems, the algorithm yields results comparable to those obtained by state-of-the-art ILP algorithms. For the KRK problem, the dataset consists of 5 training sets of a 100 examples, and one test set of 1000 examples. The background knowledge consists of 50 facts that describe the ordering of rows or columns on a chess board. The average accuracy obtained was of 93%, which is comparable to the results achieved by FOIL, while it is slightly inferior to DOGMA's. The dataset regarding the mutagenic problem consists of a set of 42 examples, 13 positive and 29 negative. The background knowledge contains 12,203 elements on atomic structure and bonding. The average accuracy obtained was of 73%. In particular, for the mutagenesis dataset, the use of the biases on both the language and the background knowledge sensibly improved the efficiency of the system, letting, however, the system achieve good results. The results indicate that knowledge based evolutionary programming provides an effective method for learning in FOL. Moreover, experiments on datasets containing noise indicate that the algorithm is capable of handling also imperfect data.

References

[Muggleton, 1999] Muggleton, S. (1999). Inductive logic programming: issues, results and the challenge of learning language in logic. *Artificial Intelligence*, 114:283–296.

Estimating Stock Price Predictability Using Genetic Programming

Minglei Duan

Dept. of Electrical and Computer Engineering
Marquette University
PO Box 1881, Milwaukee, WI 53201-1881

Richard J. Povinelli

Dept. of Electrical and Computer Engineering
Marquette University
PO Box 1881, Milwaukee, WI 53201-1881

A new method that quantifies the predictability of a stock's price is presented. This new method overcomes some problems with previous approaches to stock price predictability. The new method shows which stocks are more predictable, and hence can help in making investment decisions. Genetic Programming is used as the modeling tool. Preliminary experiments are conducted to show the advantages of this method.

Time series predictability is a measure of how well future values of a time series can be forecasted. Measuring the predictability of a time series is important because it can tell whether a time series can be predicted before making a prediction. Therefore prediction of time series with low predictability, such as a random walk time series, can be avoided. A good metric for time series predictability also provides a measure of confidence in the accuracy of a prediction. This is especially helpful to minimize the risk when making an investment decision.

An h-metric was introduced by Kaboudan [1], which measures the probability that a series is GP-predictable using Genetic Programming [2]. By design, the computed metric should approach zero for a complex signal that is badly distorted by noise. Alternatively, the computed metric should approach one for a time series with low complexity and strongly deterministic signal.

This metric is based on comparing two outcomes: the best fit model generated from a single data set before shuffling with the best fit model from the same set after shuffling. The shuffling process is done by randomly scrambling the sequence of an observed data set using Efron's bootstrap method. Specifically, the unexplained variations, which are measured by the sum of squared error before and after shuffling of a time series are compared.

While applying the h-metric to estimate stock price predictability, two main problems have been observed. First, the value of the metric largely depends on the length of the time series. The source of this effect is mainly due to the nonstationarity of stock price time series, and the nonstationarity becomes more evident as the sample size increases. The second problem is a derivation of the first one. For a long-term stock series, the value of the h-metric will be distributed in a very narrow range. Hence, it largely reduces the resolution of the metric.

These problems are resolved by using a shifting window, so that the h-metric can be estimated on the same sample size, as long as the window length is fixed. By changing the window length, the range of the metric can be adjusted. The overall predictability of a time series can be estimated by calculating the average h over all windows.

In order to test the new metric, it is applied to three different kinds of time series: a deterministic time series (Mackey-Glass map), a stock price time series, and a random walk time series. It can be seen clearly from the experiments that different kinds of times series yield significantly different results (Table 1).

Table 1: Experimental results

Time series	Predictability
Mackey-Glass	0.996
Random Walk	0.140
GE Stock	0.485

This method has been shown to be able to distinguish stock price time series and random walk time series effectively. Future work needs to be done comparing more different stocks to test whether the metric can quantify the predictability accurately. This method may be helpful in making investment decisions.

References

[1] Kaboudan, M. 1998. A GP approach to distinguish chaotic from noisy signals. *Genetic Programming 1998: Proceedings of the Third Annual Conference*, San Francisco. CA: Morgan Kaufmann, pp. 187-192

[2] Koza, John 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: The MIT Press.

[3] Kaboudan, M. Genetic Programming Prediction of Stock Prices, *Computational Economics*, to appear.

A new methodology for the Placement and Routing problem based on PADGP

F. Fernández
Centro Universitario de Mérida
Universidad de Extremadura
C/ Calvario, 4. 06800 Mérida. Spain
fcofdez@unex.es

J.M Sánchez
Escuela Politécnica
Universidad de Extremadura
Avda de la Universidad. Cáceres. Spain.

M. Tomassini
Institut d'Informatique
Université de Lausanne
Switzerland

Abstract

We present results from the application of a new methodology based on Parallel and Distributed Genetic Programming (PADGP). It allows us to automatically perform the placement and routing of circuits on reconfigurable hardware. For each of the problems we have dealt with, the methodology finds several different solutions.

1 DESIGN BASED ON FPGAS

Field Programmable gate arrays (FPGAs) are powerful devices for implementing complex digital systems. FPGAs are arrays of prefabricated logic blocks and wire segments with user-programmable logic and routing resources. When programming an FPGA, we previously obtain a circuit description via logic synthesis. We take this description and we map and convert it into the modules and routing resources available in FPGAs. Bearing in mind that both logic blocks and routing resources are predefined in an FPGA chip, circuits must be laid out within it.

2 METHODOLOGY.

As we are using GP, it is customary to codify the problem by means of trees. Each circuit to be placed and routed should thus be coded as a tree. Figure 1 graphically depicts the way circuits are encoded as trees. In [Fernández et al 2000] we can find a more detailed description of the encoding process.

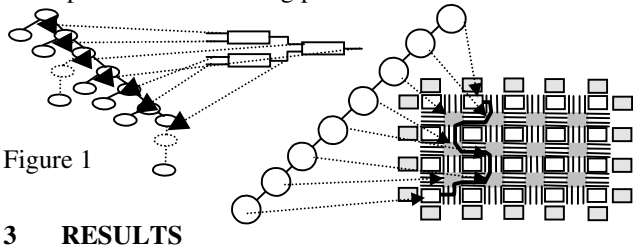


Figure 1

3 RESULTS

Figure 2 graphically depicts the circuits that we have tested to see if the methodology is capable of placing and routing circuits. Figure 3 shows one of the solutions that PADGP finds for this benchmark problem

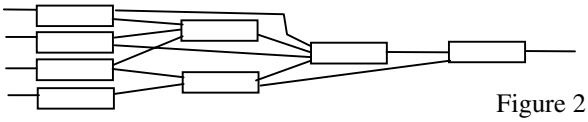


Figure 2

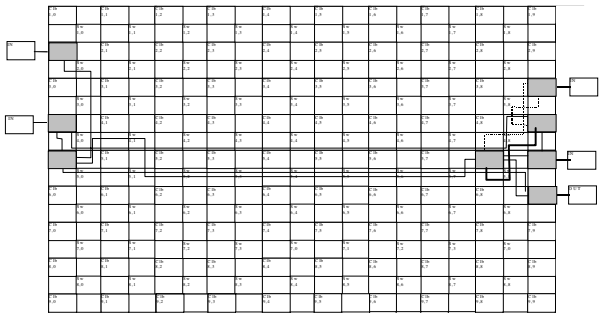


Figure 3

We have designed experiments using several populations, a migration rate of 1 individual per generation and random communication topology. We have performed several trials each time with a different number of individuals or populations. In figure 4 we can see that using 2 or 5 population we get better results than using only 1 when the total number of individuals is 500.

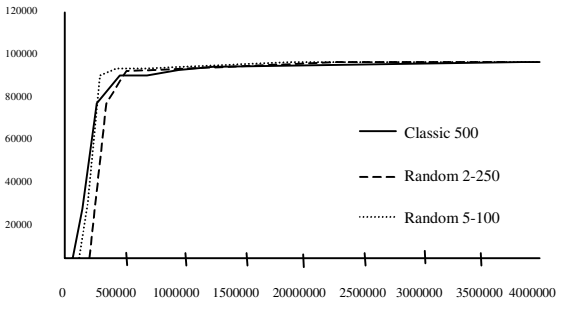


Figure 4

4 REFERENCES

[Fernández et al 2000] F. Fernández, J.M. Sánchez. M. Tomassini. Proceedings XV Design of Circuits and Integrated Systems 2000.

EvolVision – an *Evolvica* visualization tool

Tim Führer
Department of Computer Science II
University of Erlangen-Nuremberg
Martensstr. 3
D-91058 Erlangen, Germany
tim.fuehner@stud.informatik.uni-erlangen.de

Christian Jacob
Department of Computer Science
The University of Calgary
2500 University Drive N.W.
Calgary, Alberta T2N 1N4, Canada
jacob@cpsc.ucalgary.ca

1 INTRODUCTION

EvolVision is an evolutionary algorithm visualization tool, designed for the *Evolvica* framework—a *Mathematica* library, which provides notebooks and routines for a wide range of evolutionary algorithms [Jac01]. Owing to an open architecture, *EvolVision* is not limited to the *Evolvica* framework or any specific programming language or computer platform but can be used in various environments [Füh00].

The visualization tool aims at applying and exploring standard visualization techniques, being easily extendible by new (experimental) visualization techniques, and visualizing different levels of the evolutionary algorithm. It gives a coarse overview of an algorithm's behavior, displays detailed views of individuals and their properties, and provides fine grained views of the algorithm's data structures.

2 ARCHITECTURE

EvolVision is implemented in the Java programming language and designed as a client/server application. Thus the visualized program has to undergo only marginal modifications and the visualization tool has no remarkable influence on its runtime behavior. Another benefit is its independence from a specific platform or operating system.

The communication between *EvolVision* and the visualized program is realized by a protocol language, which uses a grammar similar to the *Mathematica* programming language [Wol96]. This language is also used to store and review former experiments.

EvolVision is a core program that can be adjusted to the needs of the specific visualization task. Its flexibility is achieved by a plug-in architecture that allows easy extension of the visualization features. Currently there are four plug-in categories: 1. *in-line* visualization plug-ins (i.e., visualization of individuals on the main canvas), 2. *singled out* visualization plug-ins (i.e., visualization of individuals in separate windows), 3. *population-wide* visualization plug-ins, and 4. *experiment-wide* visualization plug-ins.

3 VISUALIZATION FEATURES

EvolVision's main canvas provides an overview over the whole evolutionary algorithm. It displays the populations of each generation and the fitness value of the best individual and the average fitness. A population consists of its individuals, each of which is represented by its fitness value and a *schematic*, *genotypic*, or *phenotypic* view. All representations are implemented as plug-ins.

Another core feature of *EvolVision* is the interactive generation of pedigree diagrams. The descent of individuals is depicted on demand. In order to identify the differences between the genomes of an offspring and its parents, the effect of genetic operators can additionally be highlighted in an individual's genotypic view.

REFERENCES

[Füh00] Tim Führer. EvolVision – Visualization of Artifical Evolution in the Evolvica Framework. Studienarbeit, Department of Computer Science II, University Erlangen-Nuremberg, Erlangen, 2000. see also: `http://www.cpsc.ucalgary.ca/~jacob/Evolvica/EvolVision`.

[Jac01] Christian Jacob. *Illustrating Evolutionary Computation with Mathematica*. Morgan Kaufmann, San Francisco, CA, 2001.

[Wol96] S. Wolfram. *The Mathematica Book*. Wolfram Media/Cambridge University Press, Cambridge, MA, 3. edition, 1996.

An Engineering Approach to Evolutionary Art

J.I. van Hemert
jvhemert@cs.leidenuniv.nl
Leiden Institute for Advanced Computer Science
Leiden University, The Netherlands

M.L.M. Jansen
mjansen@cs.leidenuniv.nl

Abstract

We present a general system that evolves art on the Internet. The system runs on a server which enables it to collect information about its usage world wide; its core uses operators and representations from genetic programming. We show two types of art that can be evolved using this general system.

In evolutionary art we strive for a system that creates art using the principle of evolution: The survival of the fittest, or in this case, the survival of the most beautiful. Often this goal is achieved using an evolutionary algorithm of some form. All systems share a common feature: Human intervention to determine what is nice and what is ugly, in other words, a human fitness function.

The evolutionary algorithm used here is a crossing between a genetic program and a generic evolutionary algorithm. Most of its features are from genetic programming, but it does not share the main paradigm of creating executable material that can be applied to many different inputs. We point the reader to (van Hemert and Jansen, 2001) for all the details about the implementation of the system.

Most evolutionary art systems run on a single machine, which, in itself, is not a striking property, but the main restrictions these systems have is that they interact solely with the person behind the same machine. Even if such a system would be popular its output will not go beyond the user and her machine. Here we strive for a system that is accessible for many people at the same time, gathering information about the decisions these people make. All of this is made possible through the Internet and the common gateway interface (CGI).

The fact that the system is on-line all of the time helps us, the researchers, to get the assistance of many people. To be able to make statements on these decisions we require a large amount of data, because this type of research is based largely on subjective decisions. Normally we would have to search actively for subjects that are willing to assist in these experiments, but here they can voluntarily and anonymously visit the page on the Internet and use the system.

If we want to gain information on what the average person likes and dislikes, we need to record the actions of many visitors. Hence, we connect the system to a database that stores the paintings that have been presented to users that have visited. Using this we are slowly building up a gene bank of evolved art.

Although the gene bank of images is constantly changing we provide pieces of art that are currently ranked high.

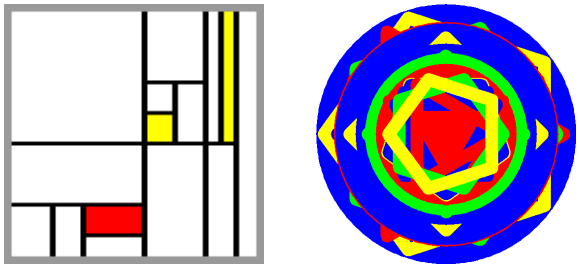


Figure 1: A Mondriaan (left) and mandala (right)

We invite the reader to visit our system on the Internet to fully experience its dynamics, it can be found at

`http://www.liacs.nl/~jvhemert/eartweb/`

References

van Hemert, J. and Jansen, M. (2001). An engineering approach to evolutionary art. Technical Report ?, Leiden Institute for Advanced Computer Science, Leiden University.

Fault-Tolerant Computing with N-Version Genetic Programming

Kosuke Imamura, James A. Foster

Initiative for Bioinformatics and Evolutionary Studies (IBEST)
Computer Science Dept. University of Idaho
Moscow, ID 83844-1010
{kosuke,foster}@cs.uidaho.edu

Abstract

We applied N-version genetic programming (NVGP) to a path prediction problem, and compared the results with a single version GP. Statistics from the experiment suggest that NVGP is a viable method to increase reliability, which reduces output variance and thereby expected meantime to system failuer.

1 INTRODUCTION

Programs inferred by sample training based methods such as genetic programming are likely to be incorrect, unless the sampling is exhaustive [1]. To cope with this dilemma, we developed an redundant module software system with an isolated island model GP, and applied it to predict the next location of a moving object. A fundamental assumption of N version programming is that independent modules will have uncorrelated faults, so that a composite system will be more reliable by avoiding simultaneous faults in different modules [2].

2 EXPERIMENT

Our NVGP consists of 5 GPs and one master. The master averages each GP's prediction at time t_i to predict the next location of a moving object at time t_{i+1}. After the prediction is output, the master receives the actual location from an external source as feedback. If a GPs prediction error is beyond a pre-determined threshold, the GP is retrained using the 5 most recent actual locations received by the master. The moving object traverses the hand made path shown in Figure 1.

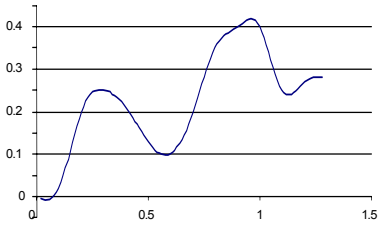
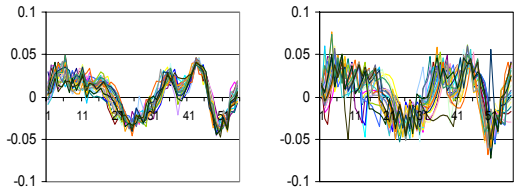


Figure 1: Path to be predicted

Figure 2 plots the prediction errors margin of NVGP and a single version GP. The plot band of NVGP is narrower than the single GP band with statistcal significance.



NVGP Single Version GP

Figure 2: Prediction Error Plot

Table 1 shows the increased reliability of NVGP over a single version measured in terms of number of prediction errors on the entire path. All the NVGP predictions use a 0.05 error threshold.

Error Threshold	Reliability Increase
0.02	22.9%
0.03	163.2%
0.04	332.0%

Table 1: Reliability Increase by NVGP

3 CONCLUSIONS

The NVGP predicts the target path with a statistically significantly narrower error band than a single GP system, even though the NVGP and the single GP produce the same mean of errors. This indicates a longer meantime to system failure, and suggests a degree of fault in the isolated island NVGP.

References

[1] Kosuke Imamura, James A. Foster and Axel Krings, "The test vector problem and limitations to evolving digital circuits", Proc. NASA/DoD Workshop on Evolvable Hardware (EH), IEEE Press, pp. 75-80, 2000

[2] Avizienis, A. and J.P.J. Kelly, "Fault Tolerance by Design Diversity: Concepts and Experiments", IEEE Computer, vol. 17 no. 8, pp. 67-80, August 1984

Network Structure Oriented Evolutionary Model–Genetic Network Programming–and Its Comparison with Genetic Programming

Hironobu Katagiri
Dept. of Electrical and Electronic Systems Eng.
Kyushu Univ., 6-10-1, Hakozaki, Higashi-ku
Fukuoka 812-8581, Japan

Kotaro Hirasawa
Dept. of Electrical and Electronic Systems Eng.
Kyushu Univ., 6-10-1, Hakozaki, Higashi-ku
Fukuoka 812-8581, Japan

Jinglu Hu
Dept. of Electrical and Electronic Systems Eng.
Kyushu Univ., 6-10-1, Hakozaki, Higashi-ku
Fukuoka 812-8581, Japan

Junichi Murata
Dept. of Electrical and Electronic Systems Eng.
Kyushu Univ., 6-10-1, Hakozaki, Higashi-ku
Fukuoka 812-8581, Japan

A large number of studies have been made on the planning of real mobile robots and virtual agents. The planning, generating the behavior sequences, is generally complex and hard to design for us. To overcome these problems, we need to make the planning without human. Many studies on automatic design of the behavior sequences, therefore, have been reported. There are many kinds of such studies and especially the evolutionary optimization techniques, such as Genetic Programming (GP) and Evolutionary Programming (EP), stand out. In this paper, the evolutionary optimization technique is also used to generate the behavior sequences for agents in virtual worlds. The distinguished point of this study is that it bases on the network structure to express the behavior sequences, while GP bases on the tree structure. To distinguish the proposed method in this paper from conventional GP, we call the proposed method Genetic Network Programming (GNP). The comparative simulations are executed using the tileworld environment, which is famous for as a test bed problem, to show the differences between GNP and GP. The tileworld is two-dimensional latticed world. There are agents, tiles, holes and barriers. In this world, the goal of these agents is to drop all the tiles into the holes.

GNP is especially developed to express the behavior sequences of the agents, and is based on the network structure. Tree based structure well describes the knowledge and rules, but it seems to be the structure consisted of static knowledge and rules. The flexible behavior may be obtained by using the past behaviors indirectly rather than by using only the current situation or using the past behaviors intentionally. Almost all the studies insisted on the determination of the agent's behavior from the root node or start node every each agent's behavior. On the contrary, GNP won't determine the agent's behavior from the specified node, that is, all behaviors are considered to be sequences. This is the main reason why GNP adopts the network structure.

GNP is composed of many and various nodes. These nodes are roughly classified into two kinds of nodes: `JUDGEMENT NODE` and `PROCESSING NODE`. The former nodes are still more classified into some kinds of concrete functional nodes, and in the same way the latter nodes are. The nodes belong to `JUDGEMENT NODE` describe some kinds of judgement, e.g. "Is there something in front of the agent?", and the nodes belong to `PROCESSING NODE` describe some kinds of action, e.g. "Move forward". The behavior sequence is described by connecting these nodes with each other. A token that shows the current node moves through the network, and the transfer rule of the token is determined by the consequent of judgement and action. Take care, the token starts from the start node at the beginning, but subsequent transfer depends only on the consequent of judgement and action. In this way, the agents behave according to the network flow.

The evolutionary operator such as crossover and mutation are used to evolve the GNP structure. These operators are applied almost the same way in Genetic Algorithm, and applied only to the connection between the nodes. In the other words, the functions of the nodes are never changed but the connections are changed.

From the simulations, although it is difficult to compare completely fair between GNP and GP because of the differences of the structure and evolutionary method, better results were achieved with GNP than with GP in three points, for this particular problem and the parameters tested. The first is the memory efficiency, the second is quick convergence and the last is the high success rate.

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

There has been a big interest in inducing classes of grammars in the area of machine learning. The symbolic nature of the grammar induction problem makes it suitable for the GP-approach. However the straightforward application of the GP on **Context Free Grammar Induction** problem fails to generate a satisfactory solution. In this paper a new approach is presented where the aim is to formalize a control module for GP which would direct the search only among well-fit grammars.

GP is an induction method used to search over a huge state space consisting of structured representations. Induction of context-free grammars can be considered as a suitable symbolic task for the GP method. The left-hand side of a rewrite rule in a grammar can be treated as a function which is composed of the right-hand side elements of the same rule. Thus, it is possible to represent context-free grammars as as structured trees. When the GP method is used in a straightforward manner for the CFG-induction problem, the output grammar induced is far away from a reasonable abstraction over the training sentences. It seems that the limitations for the convergence appear due to the nature of the problem. The interdependency among subparts of a context free grammar is high. That is, the contribution of a part of a grammar to the fitness function heavily depends on some other subparts. Hence the risk of destroying the interdependent overall structure and causing a dramatic fall of the fitness value after a genetic operation is quite high.

It can be claimed that the above problem is based on the limitations of the tree representation used in GP. The tree abstraction is capable of representing subparts of a problem and how these subparts are connected to each other. However it fails to capture the global information based on the interdependency relation existing in a grammar, hence it becomes possible to destroy this global structure throughout a genetic operation ending up with invalid offsprings that cannot parse any of the training sentences at all.

The control module proposed tries to overcome the problem by using a representation that would capture the dependency information in a grammar. In order to achieve this goal the chromosomes are transformed to single points in an n-dimensional space which will be used by the control module. It is aimed that the control module would use these atomic representations and would try to determine prototypes for the valid and invalid elements in the domain. Then, these prototypes can be feeded in the genetic search and the genetic search can use them to determine the consequences of a genetic operation beforehand and perform the right genetic operations that would keep the search only among the well-fit elements.

The transformation of the chromosomes is carried out by mapping terminal elements to base vectors and non-terminal elements to vectors obtained by adding the vectors of their arguments. On the other side, the interaction between the genetic search and the control module is formalized as follows: The genetic search is started and for each chromosome that appears, the corresponding vector representation and its fitness value are sent to the control module. After collecting a certain amount of data, the control module uses a classification algorithm and forms prototypes for the valid and invalid chromosomes. After the prototypes are determined, before each genetic operation different alternatives are analyzed and the best alternative that is the one that would produce offsprings closest to the valid prototypes is chosen.

Several trials have been carried out and it has been observed that the controlled genetic search can outperform the straight-forward application of GP on the problem.

Using Heuristics Related to Cellular Automata Behavior Forecast to Improve Genetic Search for a Grouping Task

Gina M. B. Oliveira
Univ. Presbiteriana Mackenzie,
R. da Consolação 896, 5º Andar,
São Paulo, SP - Brazil

Pedro P. B. de Oliveira
Univ. do Vale do Paraíba,
Av. Shishima Hifumi 2911,
São José dos Campos, SP - Brazil

Nizam Omar
Inst. Tecnológico de Aeronáutica,
Praça Marechal Eduardo Gomes 50,
São José dos Campos, SP - Brazil

Cellular automata (CA) are discrete complex systems which possess both a dynamic and a computational nature. Based only upon their definition, it is not possible to forecast their dynamic behavior. Various studies in the context of one-dimensional CA have been carried out on defining parameters, directly obtained from their transition rule, which have been to shown to help forecast their dynamic behavior. From the analysis of the parameters published in the literature, and also from others investigated by the authors of this paper, we selected a set of five parameters (Oliveira, 1999, 2001).

Various investigations have been carried out on the computational power of CA, with concentrated efforts in the study of 1D CA and their computational abilities. One of the approaches is the use of Genetic Algorithms (GA) to look for CA with a predefined computational behavior (Mitchell, 1994). Our approach being the use of the parameter set as an auxiliary metric to guide the GA search. We rely on a simple GA, similar to that used in (Mitchell, 1994), with the appropriate modifications.

We show here the experiments involving an adaptation of the Ordering Task (Sipper, 1998) which yielded what we named the Grouping Task (GT) (Oliveira, 1999): Let us consider an arbitrary initial configuration (IC) of a lattice formed by N_0 0s and N_1 1s, randomly distributed. The CA transition rule is said to be able to solve the GT if, after T time steps, it still has N_0 0s and N_1 1s, but distributed in a such way that from the left to the rightmost cell there can only be one transition from 1 to 0.

The environment built to evolve the GT used radius 2 CA, using a population of 100 individuals, evolving during 50 generations. Elitism was used at a rate of 20%. One-point crossover was used at a rate of 80%. Mutation was applied at a rate of 2% per bit. Each individual evaluation was obtained, at each generation, out of testing the performance of the automaton in 100 different ICs, which accounted also for partial solutions. The final efficacy of the GA run was measured by testing the performance of the best rule found in 10^4 different ICs, without considering partial solutions. The GA environment was modified so as to incorporate the parameter-based heuristic. Basically, the parameter-based heuristic is coded as a function (referred to as *Hp*), which returns a value between 0 and 100, depending on the rule's parameter values. *Hp* is then used to bias the GA operation, in the following ways. The fitness function of rule was made by the weighted average of the efficacy in 100 ICs and the function *Hp*. The reproduction was biased: in order to select the crossover point and the bits to be mutated, various attempts were made; only those that generated rules with high *Hp* value were selected.

The experiments was performed for three different sizes of lattice: 9, 19 and 49 cells. A series of GA runs was performed for each size. The introduction of the parameter-based heuristic entailed a substantial improvement in the best rule found: 29%, 73% and 119% for lattice 9, 19 and 49, respectively (Oliveira, 1999). Other computational tasks were studied in our parameter-based heuristic approach (Oliveira, 2000). However, the current experiments involving GT differ from those experiments precisely in the fact that, since it is a new task in the literature, our approach was left without any *a priori* knowledge for usage as heuristic information. In spite of that, the results were extremely encouraging, a clear suggestion that even with no previous knowledge about a task, the heuristic based on the forecast parameters alone, can still guide the GA search towards higher efficiency regions of the search space.

References

M. Mitchell, P. Hraber and J. Crutchfield (1994). Evolving Cellular Automata to Perform Computations: Mechanisms and Impediments. *Physica D*, **75**:361-391.

G.M.B. Oliveira (1999). *Dinamics and Evolution of One-Dimensional Cellular Automata.* PhD Thesis, Aeronautics Institute of Technology, SP, Brazil. (In Portuguese).

G.M.B. Oliveira, P.P.B. de Oliveira and N. Omar (2000). Evolving solutions of the density classification task in 1D cellular automata, guided by parameters that estimate their dynamic behaviour. *Artificial Life VII*, MIT Press.

G.M.B. Oliveira, P.P.B de Oliveira and N. Omar (2001).Guidelines for Dynamics-Based Parameterization of 1D Cellular Automata Rule Spaces. *Complexity,* **6** (2).

M. Sipper (1998). A simple Cellular Automata that solves the density and ordering problems. *International Journal of Modern Physics*, **9** (7).

Building a Taxonomy of Genetic Programming

Peter Martin
School of Computer Science,
Birmingham University,
Birmingham, B152TT, England
E-mail: peter.martin@marconi.com

Abstract

There is still a lack of theoretical guidance on the selection of operational parameters and only a handful of empirical studies have provided help with parameter selection, usually for a limited set of problems. By building a taxonomy of GP it is envisaged that further guidance will emerge to assist users of GP to choose appropriate parameters.

1 THE APPROACH OF THIS CONTRIBUTION

The primary aim of this work is to create a taxonomy by analysing the results of previous experiments and systems. It is then hoped that the application of the taxonomy to new problems will help workers in GP arrive at suitable parameter sets. It is noted that there is little in the way of a taxonomic analysis of GP in the literature.

This contribution starts by considering a variety of problems that have been tackled using GP. For each problem a number of attributes are collected and analyzed. In general the attributes include those which are commonly discussed in the GP literature. In addition, the problems themselves are analyzed to identify some more general attributes that may be of use when constructing the taxonomy. Once identified, the attributes can be used to construct the groups (taxa) and to separate these groups into subgroups (taxons).

A number of key characteristics of a taxonomy are identified and the resultant taxonomy evaluated against these criteria.

The attributes identified are placed into one of three high level categories. These three categories reflect the process of decomposition often encountered in problem solving and were chosen to reflect the process of solving a problem using GP.

2 A PRELIMINARY TAXONOMY

The attributes analysed so far have been used to construct a taxonomy, where the x axis represents the 3 main categories of attributes. Within each category the major attributes are placed. Where appropriate these have first been grouped together into distinct taxa, and these taxa then further separated into taxons. This taxonomy is shown in figure 1.

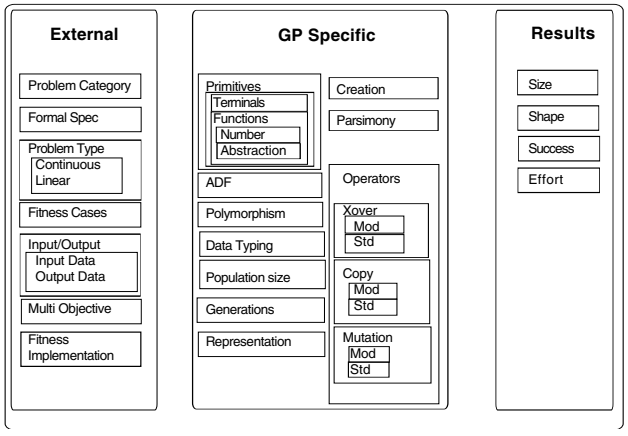


Figure 1: A Taxonomy of genetic programming

At the time of writing, over 130 examples of GP have been identified, and are being analysed, though the data set is not complete for these examples. The collection and analysis is ongoing work.

Acknowledgment

This work was sponsored by Marconi Communications Limited.

Discovering Fuzzy Classification Rules with Genetic Programming and Co-Evolution

Roberto R. F. Mendes Fabricio de B. Voznika Julio C. Nievola Alex A. Freitas

PUC-PR
PPGIA - CCET
Av. Imaculada Conceição, 1155,
Curitiba - PR, 80215-901. Brazil
{alex, nievola}@ppgia.pucpr.br
http://www.ppgia.pucpr.br/~alex
+55 41 330-1669

CEFR-Miner (Co-Evolutionary Fuzzy Rule Miner) is a data mining system for the classification task. It uses two evolutionary algorithms: a genetic programming (GP) algorithm evolving a population of fuzzy rule sets and a (1+5)-evolution strategy (ES) evolving membership function definitions. These algorithms co-evolve in order to generate a fuzzy rule set and a set of membership function definitions that are well adapted to each other.

CEFR-Miner can handle both categorical (nominal) and continuous attributes. For continuous attributes, instead of using crisp conditions such as (*Age* < *25*), our system represents these attributes using fuzzy conditions like (*Age* = *young*), where *young* is a fuzzy linguistic term. Fuzzy linguistic terms are intuitively more comprehensible to the end user, and they overcome the sudden and unnatural transition between categories associated with crisp conditions.

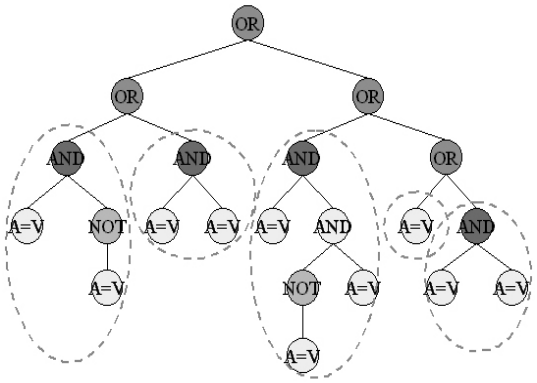


Fig. 1 GP individual with 5 rule antecedents

Each GP individual corresponds to a set of fuzzy rule antecedents (IF part) encoded in disjunctive normal form (DNF) – see e.g. Fig. 1. (All rules have the same consequent, i.e. THEN part, which does not need to be encoded in the individual.) Each rule condition represents an attribute-value pair <*A* = *V*>. If the condition is fuzzy the definitions of the membership functions required to interpret it are provided by a (1+5)-ES algorithm. Hence, the membership function definitions of the ES individual are used to evaluate the fitness of GP individuals.

The (1+5)-ES algorithm evolves a "population" of a single individual using only mutation (and not crossover). This single individual represents a set of membership functions definitions for all linguistic terms of all fuzzy attributes. The fitness of this ES individual is based on the average fitness of several GP individuals (i.e. fuzzy rule sets) interpreted with the membership function defined by the ES individual, rather on the fitness of a single GP individual. This improves the robustness of the ES's fitness evaluation.

CEFR-MINER was evaluated on five public domain datasets and the results were compared to BGP [Rouwhorst & Engelbrecht 2000] and ESIA [Liu & Kwok 2000]. Table 1 shows the accuracy rate on the test set for the three systems. (The results for CEFR-MINER were obtained by running a 5-fold cross-validation procedure. The results for BGP and ESIA were taken directly from the above references.) The numbers between brackets are standard deviations.

Table 1: Accuracy rate (on test set)

Data set	CEFR-MINER	ESIA	BGP
CRX	84.7 (±3.5)	77.39 (±0.23)	N/A
Heart (statlog)	82.2 (±7.1)	74.44 (±0.26)	N/A
Ionosphere	88.6 (±6.0)	N/A	89.2
Iris	95.3 (±7.1)	95.33 (±0.03)	94.1
Hepatitis	87.5 (±13.1)	N/A	N/A

References

J.J. Liu & J.T. Kwok (2000) An Extended genetic rule induction algorithm. *Proc. Congress on Evolutionary Computation (CEC-2000)*. IEEE.

S.E. Rouwhorst & A.P.Engelbrecht (2000) Searching the forest: using decision tree as building blocks for evolutionary search in classification. *Proc. Congress on Evolutionary Computation (CEC-2000)*. IEEE.

Evolution of Program Size in Cartesian Genetic Programming

Julian Miller
School of Computer Science, University of
Birmingham, Birmingham, B15 2TT, UK
Telephone: +44 121 414 3710
Email: j.miller@cs.bham.ac.uk

Abstract

This paper presents an empirical study of the variation of program size over time, for a form of Genetic Programming called Cartesian Genetic Programming. Two main types of Cartesian genetic programming are examined: one uses a fully connected graph, with no redundant nodes, while the other allows partial connectedness and has redundant nodes. Studies are reported here for fitness based search and for a flat fitness landscape. The variation of program size with generation does not behave in a similar way to that reported in other studies on standard Genetic Programming. Depending on the form of Cartesian genetic programming, it is found that there is either very weak program bloat or *zero bloat*. It is argued that an important factor in the analysis of the change of program length is neutral drift, and that if genotype redundancy is present, the genetic neutral drift simultaneously improves search and compresses program code.

1 DISCUSSION

The view that sees neutral drift as a causative factor in program bloat has received little attention in the literature. Programs that have varying amounts of junk code within them all have the same fitness. Evolutionary algorithms, unlike strict hillclimbers (which don't exhibit bloat), do not typically demand that promotable programs (to the next generation) have an improved fitness, thus they may accept equally good solutions (i.e. fitness neutral) or even slightly worse solutions. Consequently, if there is a mechanism that can create neutral solutions a genetic drift process will occur, particularly during periods of no fitness improvement (which is when *implicit* bloat can be at its worse). In program representations that do not distinguish genotype from phenotype (i.e standard tree-based GP) this process of drift must largely occur by the insertion of junk code. In other work [1] it has been shown that genetic drift is highly beneficial in CGP as it allows constant innovation and removes genetic stagnation. This is also observed in other systems. However genetic drift with implicit introns appears to cause stagnation and supresses constant innovation. One advantage of making a distinction between genotype and phenotype is that the exploratory nature of genetic drift can occur mainly in fitness neutral space and only occasionally affect phenotype space. This means that there is no penalty associated with the neutral exploration as it is never evaluated when the fitness of a program is calculated. The argument that program bloat provides a protective mechanism for the destructive effects of both crossover and mutation (i.e. it is a *good* thing) applies equally well to explicit redundancy. Thus one can take advantage of it in CGP without paying the penalty of evaluating it. To some extent one can see fully-connectedness as an invitation for program bloat and it is really difficult to see any virtues it may have over representations that allow explicit code redundancy.

Standard CGP (without insert/delete-node operators) has a bounded program size. However this does not seem to be a large factor in program size suppression as in a flat fitness landscape the average size of the programs is always a fraction of the maximum bound. Clearly it would be a problem in a fitness based search if the bound chosen was less than the minimum size to construct a correct program.

Experiments performed indicate that implicit intron growth is not a problem and no measures need to be taken to suppress it (at least for a challenging Boolean problem). Evidence has been provided of the unbiased nature of the mutation operator by examining the behaviour of the programs under evolution in a flat fitness landscape. The central concept of the work is that allowing unconnected program nodes is very useful and improves the effectiveness of the search without having to be evaluated in the fitness function. Such representations *benefit* from explicit introns that allow program exploration through genetic drift.

References

1. T. Yu, and J. F. Miller (2001). Neutrality and Evolvability of a Boolean Function Landscape. In J. F. Miller, M. Tomassini, P. L. Lanzi, C. Ryan W. Langdon (eds.), Genetic Programming *Fourth European Conference on Genetic Programming*. (EuroGp2001). Lecture Notes in Computer Science, Vol. 2038, pp. 204-217, Springer-Verlag.

Increasing the diversity of a population in genetic programming

Patrick Monsieurs
Expertise Center for Digital Media
Limburg University Center
Diepenbeek, Belgium

Eddy Flerackers
Expertise Center for Digital Media
Limburg University Center
Diepenbeek, Belgium

Abstract

In this work we describe a method to remove individuals from a genetic programming population that are similar to fitter individuals in the population. As a result, the diversity of the population as well as the convergence speed will be increased.

1 DETECTING IDENTICAL NODES

Genetic programming uses a variable length encoding to represent a solution for a problem. This encoding is usually a tree consisting of internal nodes and terminal nodes. To detect similarity between different trees, it is necessary to be able to detect identical subtrees.

To accomplish this, before a node is created, a test is done to check if an identical node is already present. In this case, no new node is created but the existing node is re-used instead. A terminal node can be tested for uniqueness by storing the contents of all the existing terminals in a hash-table or a sorted tree. To test a non-terminal node, a unique ID is given to every node when it is created. A non-terminal node is uniquely defined by the vector of ID' s of its children. These vectors can be stored in a sorted tree, and the uniqueness of a non-terminal node can be tested by looking if the vector is present in the sorted tree. As a result of this operation, individuals that have an identical subtree share the same node that represents this subtree.

2 CALCULATING THE DIVERSITY

The diversity of the individuals in the population is calculated after a generation has been created. Every node of every individual contains a mark field, and is initially set to a default value. First, a new mark value M is selected, that is different from the mark values of every node in the population. Next, the algorithm iterates over all the individuals of the population, ordered by fitness, starting with the fittest one. The diversity of an individual is equal to 1 – (the number of nodes marked with M)/(the total number of nodes of the individual). Then, the mark fields of all the nodes of this individual are set to M. Because nodes representing identical subtrees are shared between individuals, these shared nodes of the other individuals are also marked.

The diversity is a number between 0 and 1, where 1 means none of the individual' s nodes have been used by fitter individuals, and 0 means the entire individual has been used. If the diversity of an individual is not large enough, it will be removed from the population. A possible simple test is to remove all individuals that have a diversity that is less than a fixed constant. This approach has been tested on a symbolic regression problem, with the target function $x^5 - 2x^3 + x$. The set of terminal nodes was $\{x\}$, and the set of non-terminal nodes were {+, -, *, /}, and a population size of 300 was used. The average number of generations (averaged over 50 runs) required to solve this function without using the diversity test was 17. When the minimum diversity was set to a value between 0.38 and 0.46, the average number of generations was 10. However, when the minimum diversity became too high (> 0.47), only a few individuals had sufficient diversity to survive and the performance rapidly degraded.

The disadvantage of using a constant minimum diversity is that fit individuals can also be removed. This can be solved by setting the minimum diversity to the following value: MinDiversity(i) = i/PopulationSize, where i is the position of the individual in the sorted population (i = 0 for the best individual). In this case, fitter individuals require a lower diversity to survive than less fit individuals. Using this test, the average number of generations required to find the solution for the above symbolic regression problem was 7.7.

3 CONCLUSIONS

In this work, a method is presented that increases the speed of genetic programming by a factor of more than 2, and does not require any fixed constants.

Acknowledgements

Part of this work is funded by the NFWO (National Fund for Scientific Research), project G0083.97.

Subtree encapsulation versus ADFs in GP for parity problems

Simon C. Roberts and **Daniel Howard**
Software Evolution Centre, Room U50,
DERA Malvern, Worcestershire WR14 3PS, UK
dhoward@dera.gov.uk, Tel: 44-1684-894480

John R. Koza
Stanford University,
Los Altos, CA 94023
koza@stanford.edu

This paper compares the simple subtree encapsulation scheme introduced in [2] against ADFs for solving the even-5-parity problem.

Figure 1 shows the encapsulation procedure. A first evolution stage used multiple short GP runs to process an initial terminal set. The best run was then analysed to identify all the distinct subtrees which survived the evolution. Subtrees which were structurally distinct but which produced an identical output for all training cases were said to be *operationally equivalent*. Such subtrees were grouped and represented by the smallest constituent subtree. The evaluations for subtrees which appeared as multiple instances in the final population were then stored so that they could be used as terminal data for a second evolution stage.

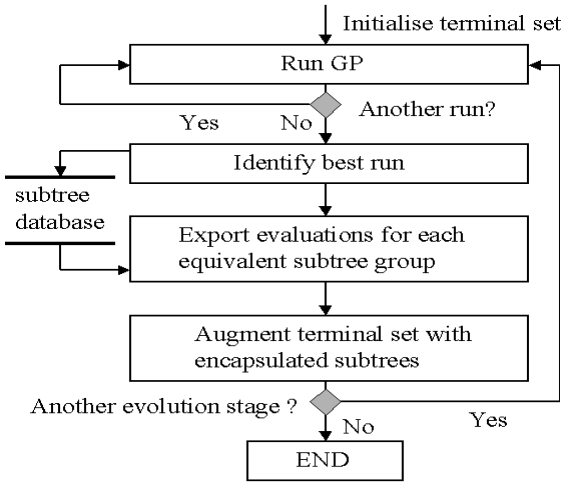


Figure 1: Subtree encapsulation procedure.

Table 1 compares different GP schemes for solving the even-5-parity problem, including the fixed architecture and the evolved architecture ADF schemes in [1]. 100 runs were conducted for each scheme and each run evolved a population of 4000 programs for 50 generations. The table gives the number of runs which solved the problem by generation 50 (N_{50}) and the estimated effort [1]. The first stage of the encapsulation procedure was evolved for 15 generations and the best run yielded 3410 distinct subtrees. Each second stage run was evolved for 35 generations using 200 randomly selected subtrees.

Table 1: Performance of different GP schemes.

scheme	N_{50}	effort ($\times$1000)
standard GP (SGP)	2	44688
200 encapsulated subtrees	45	800
1 ADF	50	672
2 ADFs	62	416
3 ADFs	59	440
evolved ADF architecture	44	864

The table shows that all modularisation schemes greatly improved upon SGP. The subtree encapsulation performed marginally better than the evolved ADF architectures but it was out-performed by the fixed ADF architectures. These comparisons demonstrate the power of modularisation, where even a simple scheme using frozen modules can contend with schemes which continually evolve modules. Analysis of the most beneficial selected subtrees could reveal an indicator for a subtree's potential usefulness. This indicator could then be used to spawn ADFs based on the best candidate subtrees.

References

[1] J. R. Koza (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press.

[2] S. C. Roberts, D. Howard and J. R. Koza (2001), Evolving modules in GP by subtree encapsulation, *Proceedings of the Fourth European Workshop in Genetic Programming*, Italy, April 2001, Springer LNCS.

No Coercion and No Prohibition - A Position Independent Encoding Scheme for Evolutionary Algorithms

Conor Ryan, Michael O'Neill and Atif Azad
Department of Computer Science and Information Systems
University of Limerick
Ireland
{Conor.Ryan|Michael.ONeill|Atif.Azad}@ul.ie

Abstract

We describe an new encoding system, *Chorus*, for grammar based Evolutionary Algorithms. This scheme is coarsely based on the manner in nature in which genes produce proteins that regulate the metabolic pathways of the cell. The phenotype is the behaviour of the cells metabolism, which corresponds to the development of the computer program in our case. In this procedure, the actual protein encoded by a gene is the same regardless of the position of the gene within the genome.

1 Introduction

The mapping from genotype to phenotype in nature is rarely as simple as the one gene-one trait methodology often employed by Evolutionary Algorithms. Moreover, the function of a gene in nature is rarely dependent on its position within the chromosome, for they usually produce the same *protein* regardless of their situation. It is the proteins produced by genes that combine to regulate the metabolism of a cell resulting in the observed phenotypic traits[1].

The function independent of location property has proved notoriously difficuly to implement in EAs. Usually, any kind of position independence forces the use of a repair operator after crossover, to ensure that every required gene is present. There can also be an issue of *overspecification*, but this is often left unrepaired, as it doesn't affect the decoding.

Having function inextricably linked to location increases the difficulty of a problem for an EA, for it is clearly more difficult for an individual to have a gene at a particular position than it is for the individual to simply possess the gene. The situation is aggravated by epistatic effects, where an increase in fitness is associated with having several correct genes, each in particular places.

There have been some attempts to design position independent Genetic Algorithms, most notably the Messy GA, but this involved a repair mechanism after crossover, and wasn't intended to deal with the evolution of grammars.

This paper presents a position independent representation that we term *Chorus*. It is so called because when the system is transcribing from genotype to phenotype there is often competition as to which protein should be dominant in regulating any one of many possible metabolic pathways that could be taken. This, we believe, is analogous to a situation where there are a number of voices striving to be heard. Typically, the loudest voice is heard, and so, the protein with the greatest *claim* to be expressed, is chosen.

2 Conclusions

We have described a new, position independent, representation scheme for Evolutionary Algorithms, termed Chorus. Chorus has been tested on two standard benchmark problems, and been shown to outperform GP on one of them.

The results indicate that, while the system does depend on the initial random seed for individuals, it appears to be robust enough to cope with relatively poor choices.

[1] Other factors are also responsible for the regulation of metabolism, our current model focuses on one of the major factors - the concentration of specific regulatory enzymes/proteins.

Genetic Programming Evolves a Human-Competitive Player for a Complex, On-line, Interactive, Multi-Player Game of Strategy

John Schloman
Department of Computer Science and Systems Analysis
Miami University, Oxford, OH 45056

Ben Blackford
Department of Computer Science and Systems Analysis
Miami University, Oxford, OH 45056

This paper describes the use of genetic programming to evolve a competitive strategy for playing Quake 2. Quake 2 is a difficult on-line game of strategy requiring the navigation of complex paths, the acquisition of various offensive and defensive weapons, and the timely use of those weapons against human-controlled or automated opponents.

Using the Q2Botcore by Ben Schwartzlander as a communications platform, our evolved individuals were able to connect to a Quake 2 server as autonomous clients. This interface limited us to only the data provided to a standard Quake 2 client (server sent game entity information and local map files).

Initial evolution involved a Q2Botcore pathing function to aid in navigation. Unfortunately inherent flaws in the algorithm led to unacceptable and suboptimal behavior. Our second attempt concentrated on omitting dependence upon pathing functions and pre-generated routing tables of nodes.

As with the classic Artificial Ant problem (Koza, 1992) we proposed that success in navigation could result from basing fitness upon resource collection. Inventory items in Quake 2 were assigned arbitrary values based upon their rarity. Inventory was evaluated every 15 seconds and individuals that reported an increase would be rewarded with another 15 seconds to continue. Individuals who were not able to "keep alive" had their fitness calculated from their inventory and exited the server. The data type for our function set was a float representing the yaw value in radians of the individual. Pitch and roll are unnecessary to navigation in Quake 2. The terminal returned the current yaw and the function set included a turn function, which operated upon it. This function set also included item targeting (returning the angle to the entity that best satisfies the visibility or distance condition), obstacle avoidance (returning a safe angle away from obstructions), and timing (allowing for decision tree branching at a given rate).

We chose a single population evolution with 200 individuals per generation and three genetic operators (reproduction, crossover, and mutation).

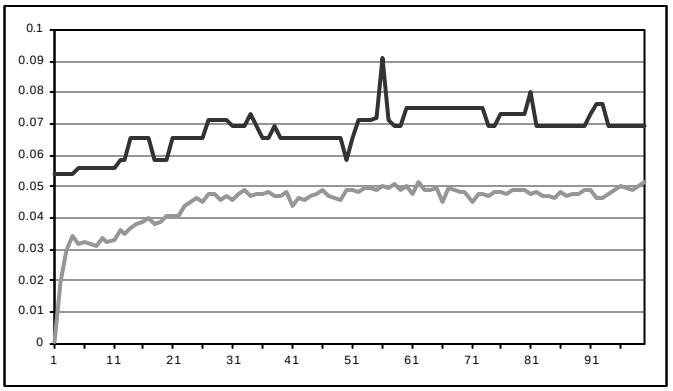


Figure 1: Maximum and Average Fitness over 100 Generations

After 100 generations, we observed efficient use of the function set for successful navigation. Early individuals began by nesting item targeting functions in turn() to successfully gather items that were not behind obstacles. Some individuals would run into walls at an angle that allowed them to continue to where another item was visible, so they would continue to collect despite the poor navigation. Later, obstacle avoidance was evolved increasing the generational averages to that of these select individuals. The average was also lowered due to negative fitness of those individuals who perished. Fitness was limited by the exhausting of local resources. The static 15-second time limit was insufficient for individuals who depleted all items to proceed to other areas of the map.

References

Koza, John R. (1992). *Genetic Programming: on the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

"Quake 2 Bot Core Homepage", Telefragged. <http://www.telefragged.com/Q2BotCore/> [Accessed 12 January 2001].

First Steps toward Automated Design of Mechatronic Systems Using Bond Graphs and Genetic Programming

Kisung Seo **Erik D. Goodman** **Ronald C. Rosenberg**

Genetic Algorithms Research and Applications Group (GARAGe) Department of Mechanical Engineering
Case Center for Computer-Aided Engineering and Manufacturing
Michigan State University, East Lansing, MI 48824
ksseo@egr.msu.edu goodman@egr.msu.edu rosenber@egr.msu.edu

1 OVERVIEW OF THE WORK

This paper suggests a method for automatically synthesizing designs for mechatronic systems. The domain of mechatronic systems includes mixtures of, for example, electrical, mechanical, hydraulic, pneumatic, and thermal components, making it difficult to design a system to meet specified performance goals with a single design tool. Bond graphs are domain independent, allow free composition, and are efficient for classification and analysis of models, allowing rapid determination of various types of acceptability or feasibility of candidate designs (Karnopp *et al*). This can sharply reduce the time needed for analysis of designs that are infeasible or otherwise unattractive. Genetic programming is well recognized as a powerful tool for open-ended search (Koza et al). The combination of these two powerful methods is therefore an appropriate target for a better system for synthesis of complex multi-domain systems. The approach described here will evolve new designs (represented as bond graphs) with ever-improving performance, in an iterative loop of synthesis, analysis, and feedback to the synthesis process.

2 BOND GRAPHS AND GP

Bond graphs consist of elements and bonds. There are several types of elements, each of which performs analogous roles across energy domains.

Bond graphs are "grown" by executing the sequence of GP functions and terminals specified by the tree (at specific bonds or nodes of the bond graph), using the embryo as the starting point. In our initial approach, the GP tree represents a program to construct a bond graph, not the bond graph *per se*. The initial studies reported here use the following set of bond graph elements: {C, I, R; 0, 1, S_f, S_e}, representing generalized capacitances, inductances, resistances, parallel and series connections, and sources of flow and effort. This set is sufficient to allow us to achieve designs that have practical meaning in engineering terms, while still permitting other methods to be used for comparison, to help to assess our initial work.

3 EXAMPLE: EIGENVALUE SEARCH

In this case, a set of target eigenvalues is given and a bond graph model with those eigenvalues is desired. The embryo model used is shown in Figure 1. The dotted box represents the initial modifiable site ("writehead"). The construction steps specified in the GP tree are executed at that point. The numbers in parentheses represent the parameter values specified (or found) for the elements.

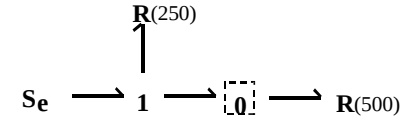


Figure 1: Embryo bond graph model

As a proof of concept for this approach, evolution of a limited set of bond graphs with specified target eigenvalues was tested, with good results produced rapidly using very limited computing resources. This provides some support for the conjecture that much more complex multi-domain systems with much more detailed performance specifications can be automatically designed, given longer execution times and/or using inexpensive cluster computing facilities.

Acknowledgment
The authors gratefully acknowledge the support of the National Science Foundation through grant DMI 0084934, beginning in August, 2000.

References

D. C. Karnopp, R. C. Rosenberg,, D. L. Margolis (2000) *System Dynamics, A Unified Approach*, 3rd ed., John Wiley & Sons

J. R. Koza, F. H. Bennett, D. Andre, M. A. Keane, F. Dunlap (1997), "Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming," *IEEE Trans. Evol. Computation.*, *1*(2), pp.109-128.

Function Sets in Genetic Programming

Terence Soule
Department of Computer Science
University of Idaho
Moscow, ID 83844
Email: tsoule@cs.uidaho.edu
Phone: 208-885-7789

Robert B. Heckendorn
Department of Computer Science
University of Idaho
Moscow, ID 83844
Email: heckendo@cs.uidaho.edu

Abstract

This research examined how the function set influences performance in genetic programming. We find that performance can vary widely with different "appropriate" function sets. In general, functions that are likely to produce solutions with very low fitness degrade the results and are heavily selected against even when the function could produce very simple solutions.

1 Introduction

There are three general guidelines used for choosing function and terminal sets in genetic programming(GP). Sufficiency, the sets must be sufficient to solve the problem. Parsimony, the sets should not contain too many extraneous functions or terminals. Closure, the functions should be able to gracefully handle all possible inputs. All three of these guild-lines were presented in Koza's original text on GP (Koza, 92). Since that time very little research has been devoted to refining those guild-lines. We examine different functions sets for the linear regression problem to begin to understand how function sets influence the search process.

2 The Experiment

The target function is a sine function (-2π, 2π). Three "reasonable" function sets are compared: F1 = {+, -, *, $\sqrt{|\,|}$}, F2 = {+, -, *, $\sqrt{|\,|}$, / }, and F3 = {+, -, *, $\sqrt{|\,|}$, /, tangent}. The divide is protected. The terminal set consists of the input variable (x) and constants in the range (-1,1). We used a generational GP, with tournament selection (3), populations size 800, 50 generations, 90/10 crossover, and using ramped half and half initial generation.

F1 acts as the control set. It is sufficient to produce a Taylor series approximation of sine ($sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - ...$). F2 is includes divide, which is commonly used in regression problems. F3 allows for a very simple, exact solution to the problem, $sin(x) = \frac{tan(x)}{\sqrt{1+tan^2(x)}}$.

Table 1: Results

Function set	Avg. Best Fitness
F1	8.04
F2	7.39
F3	9.14

3 Results

Table 1 shows the error of the best programs averaged across 25 trials. (The differences between any pair are significant (Student's two-tailed t-test $P < 0.05$) for all generations after generation 44.) Interstingly, for the average programs (not shown) F1 significantly outperformed F2 which significantly outperformed F3 (Student's two-tailed t-test $P < 0.05$). In addition, we measure the frequency of each function. Multiply and square root tend to increase, whereas divide and tangent decrease. This seemes to reflect their usefulness in generating successful programs *on average*.

4 Conclusions

The primary result is that different reasonable function sets can have a significant effect on performance. E.g. tangent performed quite poorly, even though it can create a very simple solution. Interestingly the effect of a function set was different for optimal versus average performance. We believe that these results can be partially explained by the tendency of divide and tangent to produce results with very large errors near the values where they are undefined, which will often produce large errors for program incorporating those functions.

Our results also showed that a GP selects against poor functions. However, this selection is driven by average, not optimal, individuals. These results confirm that better information regarding how to choose function sets could significantly improve GP performance.

References

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

The Differences between Social and Individual Learning on the Time Series Properties: The Approach Based on Genetic Programming

Chia-Hsuan Yeh
Department of Information Management
I-Shou University
Kaohsiung County, Taiwan 84008
E-mail: spockyeh@ms38.hinet.net

Shu-Heng Chen
Department of Economics
National Chengchi University
Taipei, Taiwan 11623
E-mail: chchen@nccu.edu.tw

Economic simulations have been widely employed in the study of economics. The learning behavior of economic agents is also modeled by the computational framework. Different learning algorithms are used to *simulate* different styles of learning behavior. In the literature, they can be classified into social learning and individual learning. In the social learning, traders learn from other traders' experience, while they learn from their own experience in the individual learning. The implications between social learning and individual learning have been stressed. However, what's more important is that the simulation result is not only influenced by *how we learn*, but also *what we can learn*. In other words, both of the learning styles and potential knowledge space contribute to the outcome. In terms of the techniques of evolutionary computation, the potential knowledge space is related to the representation, therefore, how we represent the knowledge is one of the most important steps in the economic simulation. Genetic Programming serves this purpose better than other techniques, it is then employed in this paper.

The topic used to investigate the influence of social learning and individual learning is the *artificial stock market* which has been a typical framework used to understand the behavioral foundation of finance. The dynamics of market is determined by an interaction of many heterogeneous agents. Each of them, based on his forecast of the future, maximizes his expected utility. There are two assets available for traders to invest. One is the riskless interest-bearing asset called *money*, and the other is the risky asset known as the *stock*. Traders can obtain riskless interest and cash dividends by holding money and stock respectively at each period. Dividends can follow *a stochastic process* not known to traders. Therefore, traders have to decide how much money and how many shares of stock they would like to hold at each period.

The simulation results reveal different time series properties for different learning styles. First, the purpose of social learning used in this paper is to try to figure out the regularity of price movement in order to make an accurate forecast, therefore, it is more prudent. On the other hand, traders only focus on one-period profit in the individual learning, they never care about the regularity in the past history, in the sense, they are too myopic and speculative. They use the available information to make a *better* forecast not to extract the regularity in the past history. What are the *better* strategies? The key point here is that if our forecast is based on the regularity in the past history, and the forecast is accurate, then we can make reasonable profits. This is what the social learning defined in this paper does. But, it is only a sufficient condition. The traders who follow individual learning care about one-period profit based on their forecasts. The performance of their strategies is purely determined by profit not forecasting accuracy. Therefore, for example, a trader may *forecast* that the price in next period is $100, and he decides to buy one share of stock. If the realized price in the next period is $105, then he gets more profits. In this case, the performance of this strategy in forecasting accuracy is low, while it is a *good* strategy based on the criterion of profit *under the validation process*. This situation allows more room for those strategies which don't forecast well surviving, and then, it makes them more difficult to figure out the fundamental price. Therefore, the price dynamics in the social learning is more closer to the fundamental price. In the individual learning, traders tend to overestimate the *intrinsic* value of the stock in this paper, it further induces the higher standard deviation of the stock price. Second, because there is no private information in the social learning, traders easily have similar expectations about the future. Once this situation happens, traders' behavior tends toward unification, buy or sell together. This makes the volatility of stock return higher than that in the individual learning. Based on the same argument, it is expected that the stock is not easily traded between market participants in the social learning. In other words, the stock liquidity is lower. It is also evidenced on the lower variance of the holding shares of stock.