Pathways into Genetic Programming

Michael A. Lones and Andy M. Tyrrell
Bio-Inspired and Bio-Medical Engineering
Department of Electronics
University of York
Heslington, York, England
{Michael.Lones, Andy.Tyrrell}@bioinspired.com

Abstract

Biochemical pathways are the fundamental structures of biological representations. Biological representations are the fundamental targets of natural evolution. Evolution is the fundamental principle behind genetic programming. Could biological representations be useful to genetic programming? Are biochemical pathways a suitable representation for programs? These are the fundamental questions addressed by this paper.

1 INTRODUCTION

Evolution, unlike design, takes advantage of any available behaviours. It is not limited by the cognitive capacities of a creator — it is, as Richard Dawkins says, blind. To place constraints: understandability, decomposability, synchronicity, linearity; on the process might make it seem less blind, more comprehensible; yet equally, it would be like placing walls in front of a blind person, limiting their movement, pushing them away from potentially productive pathways.

Genetic programming (GP) is a computational method that uses evolution, rather than design, to generate programs. A conspicuous trend in GP is towards increasing convergence between its products and those of manual programming. This has a number of advantages; it leads to programs that are readable, understandable, decomposed into reusable functions and, ultimately, capable of complementing the traditional software engineering process. The research reported here, however, is towards convergence not with manual programming but with natural evolution.

The justification for this is two-fold. One, there is a danger that using the tools, languages and artifacts of manual programming could be placing limiting constraints upon evolutionary search. These representations were not designed for evolution and consequently would not be expected to be evolvable. Problems such as bloat and the ineffectiveness of crossover seem to exemplify this. Secondly, and conversely, biology is a product of evolution. Therefore, its representations, its products; its very nature, are examples of approaches that work for evolution. The argument then follows that what is good for evolution may also be good for genetic programming.

2 REPRESENTATIONS

The need for a good representation in evolutionary computation, and in artificial intelligence more generally, is called the *representation problem*. Genetic programming has two forms of representation; the variational and the generative. The variational representation is a static description of a program and is subject to evolutionary variation. The main requirement for a variational representation is evolvability: the evolution of programs of increasing fitness on a generational basis when subjected to genetic variation. The generative representation is a product of the variational representation, and describes the dynamic form of a program. Its main requirement is that it can be executed. Yet, despite the different requirements of variational and generative representations, most GP systems do not distinguish between the two.

2.1 BIOLOGICAL REPRESENTATIONS

Biology does distinguish between variational and generative representations. They are called, respectively, the genetic and the phenotypic. The genetic representation, from a reductionist viewpoint, is a linear, spatially distributed, sequence of heritable attributes. Each heritable attribute describes the amino acid se-

quence of a protein. Development interprets these descriptions and generates proteins; the fundamental components of the phenotypic representation.

A group of proteins working upon a common task is called a biochemical pathway. The tasks carried out by biochemical pathways fall into three broad classes: metabolic, signalling and gene expression. Of these, metabolic pathways are considered the most fundamental for they implement the processing behaviours of the cell, whilst signalling and gene expression pathways take on a configurational role.

Biochemical processing amounts to the manipulation of a cell's chemical state through systems of chemical reactions. Metabolic pathways are composed of enzymes, a group of proteins that carry out catalytic behaviours; enabling reactions that would otherwise not be possible in the relatively low cellular temperatures. Enzymes achieve their catalytic behaviour by binding to specific chemicals, the enzyme's substrates, and guiding their reaction. Cooperation within metabolic pathways emerges from product-substrate sharing between enzymes, where the product of one enzyme becomes the substrate of another.

2.2 BIOMIMETIC REPRESENTATIONS

Biological representations possess a number of qualities conceivably useful to, but not usually found, in genetic programming representations. These, include: the specialisation of evolutionary and executable forms; evolvable representations, 'designed' for evolution; neutrality, increasing genetic diversity and adaptability; less constrained behaviour, giving more freedom to evolution; and positional independence, not limiting gene function to gene position.

A number of GP systems mimic the genetic representation of biology. Many of these have introduced a developmental stage, allowing the genetic representation to be independent of the executable representation. This has been shown to increase genetic diversity [1] and encourage neutrality [2]. A number of these approaches also allow positionally-independent genic units within the genome [3]. Other researchers have used genetic-like representations without deliberately attempting biomimicry [4, 5].

The mimicry of phenotypic representations is less common. However, computational idioms have been used to describe the action of enzymes [6] and biochemical pathways [7]. Analogues of enzyme activity have been used for computational purposes in the artificial domain [8]. Evolutionary models of pathway development have also been attempted [7].

3 ENZYME GENETIC PROGRAMMING

Enzyme genetic programming [9] is a system that mimics biology in both genetic and phenotypic representations. Phenotypic representation is based upon an abstraction of metabolic pathways. The aim of the system is to evolve analogues of metabolic pathways within combinational logic circuits. Other work in the evolution of digital circuits is reported in [10].

3.1 REPRESENTATION

Figure 1 shows the relationship between the representations of enzyme GP. During evolution, circuits are encoded as linear sequences of 'genes'; where each gene describes the input preferences, the specificities, of a particular logic gate or output terminal. A specificity is a floating point value between zero and one which indicates relative preference for inputs (substrates). Each input-consuming activity has a specificity defined for the products of every output activity.

The phenotypic representation generated by a genotype is visualised in the center of figure 1. Line weights indicate relative specificities. In practice, the network should be fully connected. However, for clarity only the dominant and a few of the recessive specificities are shown. When the circuit is realised, the dominant specificities should map to circuit connections. However, combinational circuits must be non-recurrent and consequently it will not always be possible to express a circuit element's most preferred connections. This approach is taken rather than allowing invalid circuits or constraining the genetic representation.

3.2 APPLICATION

Circuit genotypes are evolved using a diffusion-model distributed genetic algorithm. Fitness is measured by simulating the circuit against all input combinations and comparing outputs with those in the problem specification's truth table.

Population size	Generations		Successful runs	Average suboptimum	Computational effort
	Mean	C.Mean			
196	69	196	35%	95.6%	213248
324	70	127	55%	96.4%	252720
400	56	93	60%	96.2%	179200

Table 1. Results for two-bit multiplier

The method has been applied to two problems: the full adder and the two-bit multiplier. Results are shown in figure 2 and table 1 respectively. Cumulative mean

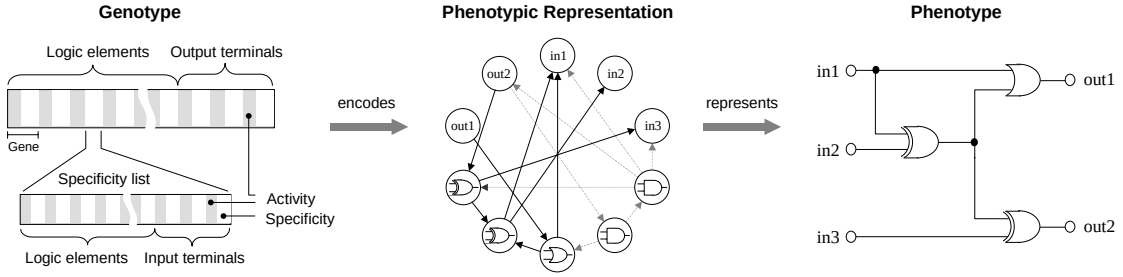

Figure 1. Circuit Representations

measures the expected wait between optimal solutions for a series of runs. Computational effort [11] measures the number of circuit evaluations required to guarantee a 99% certainty of success. For the adder, the minimum computational effort was about 42,000. Values for the multiplier are given for comparison against [10]; where results between 210,000 and 585,000 are given for different formulations of the problem. The two approaches cannot be directly compared due to differing constraints on the search space, though the performance of enzyme GP appears at least no worse.

3.3 EVOLUTION

Figure 3 shows a typical evolution of a full adder.

Neutrality

The evolutionary tree shows that most mutations are synonymous, having no effect at the circuit level. Neutral mutations fall into three classes: synonymous mutations upon recessive specificities, synonymous mutations upon dominant specificities (changing their value but not their dominance) and non-synonymous mutations that do not change the circuit's fitness. Having relatively few output bits, the full adder problem has only three effective fitness levels above those of first-generation random solutions. This both encourages non-synonymous mutations and indicates the importance of the diversity produced by synonymous mutations.

Only occasionally do mutations lead to a circuit-level change. However, when this change does occur its effect can be substantial; as the transition between circuits A and D in the diagram illustrates. Often, a circuit changing mutation occurs after a sequence of synonymous mutations; a neutral walk. An example of this occurs between the last crossover and the finding of the optimal solution.

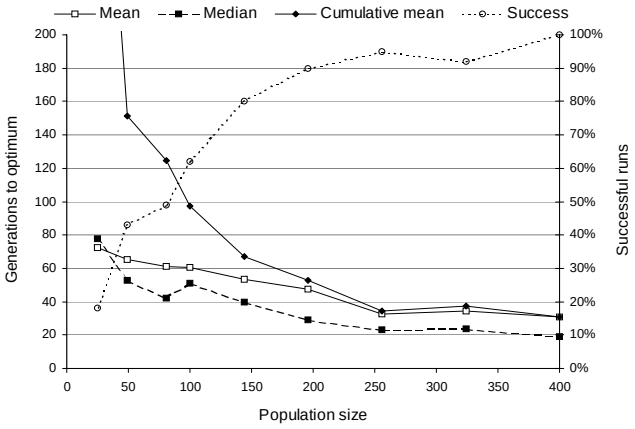

Figure 2. Results for full adder

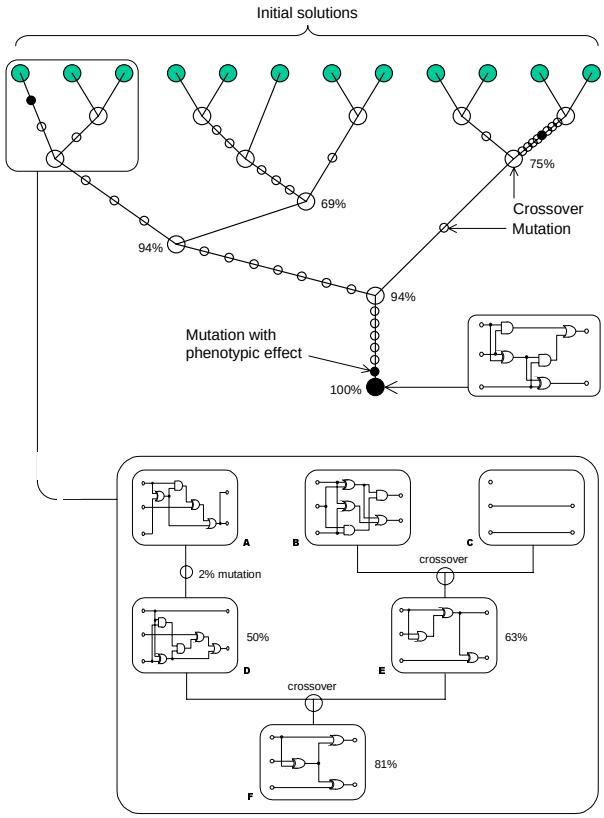

Figure 3. Evolution of a full adder

Dominance hierarchies

The effect of uniform crossover at the circuit-level, as figure 3 testifies, is also quite complex. Circuit F, for example, expresses an OR gate seen in neither parent. This is the result of a change in the dominance hierarchy, with a recessive specificity being expressed when more dominant specificities are lost. As an example, consider the specificity lists $< 7, 3, \underline{8}, \underline{9}, 1 >$ and $< 5, \underline{9}, 3, 4, \underline{8} >$. Placing crossover points after positions one and four produces a child $< \underline{7}, \underline{9}, 3, 4, 1 >$. The dominant specificities, which map to connections, are underlined. For the child solution this includes an input dominant in neither parent.

Multiplicity

From an evolutionary perspective, the potential for expression of recessive elements implies that solution genotypes have multiplicity, representing not one but many related solutions. This is best understood from a schema viewpoint. Each genotype carries both dominant and recessive schema. Crossover adds and removes schema, truncating or extending the dominance hierarchy, whilst mutation changes the relative dominance of schema, restructuring the dominance hierarchy. The advantage of multiplicity is that it groups related information, reduces computational effort and potentially makes evolution more efficient. A potential disadvantage is that, since fitness is measured solely on dominant schema and not on recessive schema, fitness values may not reflect true evolutionary values. The requirement that every consuming element has specificity for every producing element, for example, introduces a high proportion of recessive characters into the genetic representation. This may be promoting the hitchhiking of low-fitness recessive schema within high-fitness genotypes.

4 Conclusions

Neutrality, dominance hierarchies and multiplicity are all facets of natural evolution and its products. To this extent enzyme genetic programming succeeds at its aim of biomimicry. The use of an enzyme-like representation for circuit elements, and consequently a pathway-like representation for circuits, illustrates that biological phenotypic representations can be annealed to the artificial domain. Performance-wise, the method is competitive with existing methods, though it has yet to demonstrate a provable performance advantage. Partly this is due to limitations of the results, though it may also be due to deleterious simplifications found within the initial system.

References

[1] R. E. Keller and W. Banzhaf. Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes. In J. Koza et al, editor, *Genetic Programming 1996: Proceedings of the First Annual Conference*. MIT Press, 1996.

[2] C. Ryan, J. J. Collins, and M. O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In W. Banzhaf et al, editor, *First European Workshop on Genetic Programming*, volume 1391 of *Lecture Notes in Computer Science*. Springer, April 1998.

[3] S. Luke, S. Hamahashi, and H. Kitano. "Genetic" Programming. In W. Banzhaf et al, editor, *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, 1999.

[4] P. Angeline. Multiple interacting programs: A representation for evolving complex behaviors. *Cybernetics and Systems*, 29(8):779–806, 1998.

[5] J. Miller and P. Thomson. Cartesian genetic programming. In R. Poli et al, editor, *Third European Conference on Genetic Programming*, volume 1802 of *Lecture Notes in Computer Science*. Springer, 2000.

[6] M. J. Fisher, R. C. Paton, and K. Matsuno. Intracellular signalling proteins as 'smart' agents in parallel distributed processes. *BioSystems*, 50:159–171, 1999.

[7] D. Bray. Protein molecules as computational elements in living cells. *Nature*, 376:307–312, 1995.

[8] M. Shackleton and C. Winter. A computational architecture based on cellular processing. In *Proceedings of the International conference on Information Processing in Cells and Tissues (IPCAT)*, 1997.

[9] M. A. Lones and A. M. Tyrrell. Enzyme genetic programming. Accepted for the Congress on Evolutionary Computation 2001, May 2001.

[10] J. F. Miller, D. Job, and V. K. Vassilev. Principles in the evolutionary design of digital circuits — part I. *Genetic Programming and Evolvable Machines*, 1:7–36, April 2000.

[11] John Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, 1992.