Enzyme 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- The work reported in this paper follows from the hypothesis that better performance in certain domains of artificial evolution can be achieved by adhering more closely to the features that make natural evolution effective within biological systems. An important issue in evolutionary computation is the choice of solution representation. Genetic programming, whilst borrowing from biology in the evolutionary axis of behaviour, remains firmly rooted in the artificial domain with its use of a parse tree representation. Following concerns that this approach does not encourage solution evolvability, this paper presents an alternative method modelled upon representations used by biology. Early results are encouraging; demonstrating that the method is competitive when applied to problems in the area of combinatorial circuit design.

1 Introduction

Difficulty in evolutionary algorithms (EAs) is a combination of both the fitness function and the solution representation. These, in tandem with the operators used to introduce variability, determine the format of the search space traversed by the evolutionary process[1].

Representations in EAs are both generative — solution encoding — and variational[2] — subject to evolution. These introduce different, often opposing, influences on the requirements for a good representation. For instance, to meet generative demands, a representation must be efficiently executable as, for instance, is the conventional tree-structure of GP; which allows execution by a simple depth-first traversal. Variational concerns, however, require that a representation supports effective manipulation by genetic operators. The tree-structure of GP has a poor response to crossover[3], meaning that most evolutionary progress is through mutation; and suggesting that parse trees are not good variational representations.

One way of solving these opposing constraints is to introduce a duality to the representation; allowing it to have one form during evolution and another during execution. This is achieved by introducing a mapping, in the form of a developmental process, between genotype; the evolutionary representation, and phenotype; the executable representation. This mapping amounts to a separation of concerns, allowing evolution to act upon a representation suited to evolution whilst allowing the development of a solution suitable for execution. Examples of this approach include work by Keller and Banzhaf[4] and Shipman et al[5].

The need for a good representation in evolutionary computation, and in artificial intelligence more generally, is called the *representation problem*. For evolutionary algorithms, the desired representation is one that captures the property of evolvability; the ability, when subjected to evolutionary operators, to create offspring of increasing fitness on a generational basis. There have been numerous approaches to finding an evolvable representation in GP, including work by Altenberg[6], Angeline's MIPs nets[7] and Ryan et al's grammatical evolution[8]. Many of these techniques appeal to alternative computational models. However, the approach condoned in this paper is in the direction of increasing bio-inspiration; towards a system that, through greater faithfulness to biological systems, hopes to gain those behaviours found native in biology and desired in artificial systems.

With this aim in mind, the paper begins with a review of the mechanisms by which biology has solved the representation problem. Following this is a review of existing techniques in GP which have taken a bio-inspired approach, concluded by a review of the work done in this research.

2 Representation in Biology

The mapping from genotype to phenotype in nature equates to the development of a physical form from a DNA-encoded description. Before the low-level functioning of biological organism's came to be understood, the predominant view of this process was that sperm delivered a minute, yet otherwise complete, version of the organism that would grow in the mother's womb and ultimately be born. In the light of further discovery, this view gave way to the Aristotlian concept of epigenesis; the idea that embryonic development is a process of constructing a body from a blueprint. However, we now know that rather than being the deterministic process envisioned by Aristotle, epigenesis — combined with ontogeny — forms a complex, at times chaotic, system which emerges from low-level interactions between proteins formed by the body and environmentally-induced chemicals.

An organism's genome, the sum of all the DNA from all of its chromosomes, encodes the organism's proteome[9] — the set of all protein species capable of being created by the organism through the processes of DNA transcription and translation (see figure 1). Each protein is specified by a gene, a sequence of codons stating the order and nature of the com-

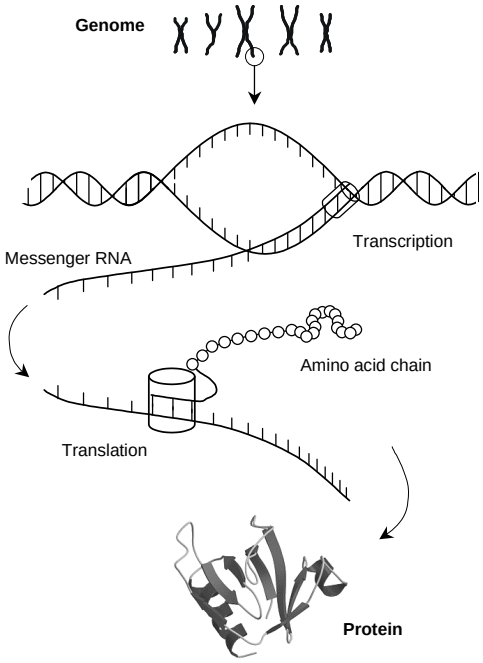


Figure 1: Genotype-phenotype mapping in biology

ponents from which the protein is to be assembled, concluded by a termination codon, marking the end of the gene. Proteins have four layers of structure. Unfolded, they form a string of amino acids. Amino acids are chemicals unified by a common structure and differentiated by the chemical properties of their side chains. However, proteins do not remain unfolded. Secondary structure, the emergence of physical members and chemical surfaces, is caused by chemical attractions between the amino acids within the chain. This effect is compounded by the aqueous environment of the cell which, through interactions between the protein and water molecules, forces the structure into a distinct folded form known as the tertiary structure. The final layer of structure, quaternary, emerges from single-chain molecules bonding together to form macromolecular complexes.

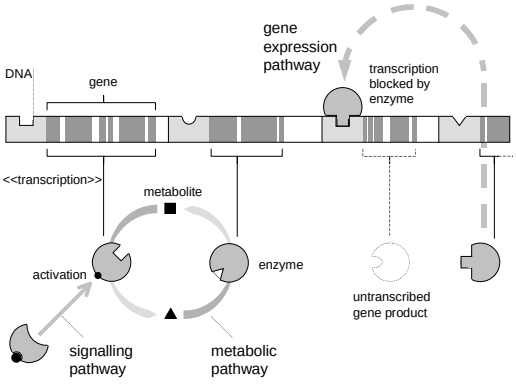
The overall behaviour of an organism does not emerge from isolated properties of individual proteins, but rather from the interactions between proteins and proteins and between proteins and other chemicals. These interactions lead to the formation of structures called biochemical pathways; sequences of protein-mediated reactions and interactions that carry out functional tasks within the organism. There are three broad categories of biochemical pathway, illustrated conceptually in figure 2. Metabolic pathways exist within cells and emerge from the interactions of locally-transcribed proteins. Signalling pathways comprise cellular responses to inter-cellular signals. Gene expression networks describe regulatory interactions between genes and gene-products. In [10], these activities are described, respectively, as *self-organising*, *self-reshaping* and *self-modification*. Self-organisation is the ability of a distributed system to carry out a unified task. Self-reshaping is the ability of an existing system to carry out multiple tasks to satisfy varying needs. Self-modification is the ability of a system to change its constitution in order to solve unforseen problems. Interactions between the three classes of pathway unify these activities and bring out the emergent behaviour of the whole.

Whilst all three classes of biochemical pathway can be considered fundamental to the operation of the organism, the most fundamental, from a reductionist viewpoint, is the metabolic network. The metabolic network carries out 'processing' functions whilst signalling and expression pathways take on a configurational role — adapting the metabolic networks to meet the current needs of the organism. The processing performed by the metabolic network amounts to the temporal and spatial organisation and orchestration of chemical reactions in order to convert chemicals available to cells into chemicals needed by cells.

For a chemical reaction to take place, an activation energy must be met. Usually, this is provided by the kinetic energy of the reactants. However, due to the relatively low temperatures found in biological organisms, the energy required for many reactions is not available. Biology solves this problem through the catalytic behaviour of a group of proteins called enzymes. Enzymes possess specificity, an ability to recognise only certain chemicals and bind exclusively to these. The chemicals recognised, the substrates, are the reactants needed for the reaction. By bringing these together, the activation energy of the reaction is reduced; making the reaction possible where it would not otherwise occur.

Metabolic pathways emerge from interactions between enzymes. A metabolic pathway consists of two interdependent flows — *reaction* pathways and *control* feedbacks. Reaction pathways are composed of systems of enzymes with tightly linked specificities for one another's products and substrates. In many cases, these pathways are forked, with more than one enzyme having specificity for a given product. Pathways may also feed back into themselves, producing an iterative structure. Control feedbacks emerge from another property of enzymes, the ability to be regulated by the binding of regulatory molecules that either enable or disable the binding of

Figure 2: Biochemical Pathways



substrates. These regulatory molecules are either products of other reactions in the pathway or are introduced by external metabolic or signalling pathways.

3 Computation in Biology

A developing view in the field of biological computation is that proteins, and enzymes in particular, have computational abilities of their own and, when organised into protein systems, act out high-level computational behaviours. Computational mechanisms of enzymes are discussed in [11]. In [12] and [13], insight is provided into the computational behaviours of enzyme systems. Bray, in particular, has explored the emergence of these behaviours in the context of simulated evolution. In [14], the computational nature of enzyme networks is highlighted by modelling them as petri nets.

Shackleton[15] has appealed to the computational aspects of enzyme systems by successfully applying artificial enzymes to an information processing task. The aim is to sort lists of numbers. Representing these lists as polymers of data items, the system simulates the action of enzyme species which either break or join lists depending upon properties of the data. Shackleton also conjectures applying genetic algorithms to the design of artificial enzyme-based systems; with enzymes being represented as sequences of codons which translate to elements of a primitive function set — constructing artificial enzymes in a process akin to protein folding. The idea of an artificial protein's computational properties emerging from the spatial ordering of primitive, amino-acid like, computational elements is further explored in [16].

4 Biology in Genetic Programming

From a procedural view, genetic programming amounts to the application of genetic algorithms to the evolution of executable structures. As such, it would be incorrect to say that genetic programming lacks biological inspiration — after all, the genetic algorithm (and genetic programming by descent) is named for its source of inspiration. However, this inspiration does not typically extend to GP's problem domain; programming.

There are exceptions. The use of a genotype-phenotype mapping, discussed earlier, allows the program's genetic representation to be improved with respect to evolutionary performance without effecting the representation used for the executable structure. However, genotype and phenotype are not independent of one another, and there is a limit to the impact allowed by this kind of tweaking. Other research has introduced alternative executable representations that implicitly provide a more evolvable genetic representation. Interestingly, some of these representations, whilst not intentionally biomimetic, do aim to provide some properties native to computation in biology. An example of this is Angeline's MIPs nets[7]. MIPs nets, named for their structural similarity to neural networks, are systems of interconnected, yet independent, parse trees. Each parse tree describes an equation, the value of which is available as an input to each of the other equations. The most important property of this system, which leads to greater evolvability, is the increased independence of functional components through distribution and the resulting limits on between-component connectivity. This property is also true of biological systems, where genes (and their products) have high internal cohesion yet low external linkage.

In other research, the representation issue has been broached from an intentionally biological perspective, typically appealing to gene expression; which Kargupta[17] describes as "the missing link in evolutionary computation". In [18], programs are represented as linear chromosomes comprising an array of parse tree 'genes'. Whilst the gene products do not truly interact (they are composed by a predefined function), the model is reported to perform competitively with existing methodologies. In [19], Luke uses a gene-based model to represent finite state automata. An automaton's genome is expressed as a multiset of genes; each of which corresponds to a state. Associated with each gene is a pattern, which specifies the 'active site' of the gene product. During evaluation, state transitions emerge from pattern matching between the patterns of the 'transcribed' states.

5 Enzyme Genetic Programming

Convention in genetic programming almost universally accepts that programs are represented as parse trees. To an extent, this dominance disguises the fact that parse trees, and trees in general, are not the only possible representations for evolving programs. Moreover, pragmatic and theoretical experience suggests that they are not good evolutionary representations. Apart from a poor response to crossover, parse trees also promote characteristics such as tight coupling, positional sensitivity and strictly linear behaviour; all of which either hinder evolution or limit the search space which it can explore. However, perhaps the most compelling argument against a representation which can be considered an artifact of human design is that it was not designed for evolution and, considering experience from the field of genetic algorithms where choice of representation is an integral part of problem solving, there is no reason to expect it to be evolvable.

The philosophy behind enzyme genetic programming is that biology has already solved the problem of an evolvable representation for programs. Of course, this is a strong assertion. For one, there is no guarantee that the representations in biology are near optimal. Secondly, we cannot, at this point, be certain that a biological representation would be suitable in artificial programmed systems. However, being a product of evolution itself, and having remained extant in the face of competing representations, the biological representation can be considered provenly evolvable. It is also evident that biology is capable of using this representation to create organisms of immense complexity. Furthermore, it has been shown that biological systems can be mapped to many

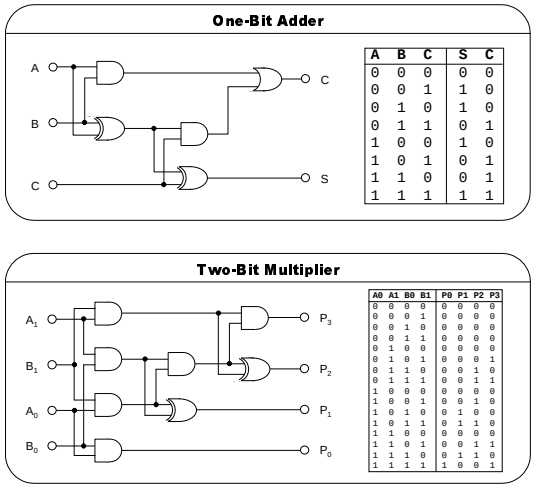
computational metaphors; for instance, information processing, pattern recognition and classification, distributed computation, agent ecologies and massively parallel computing — and successful application of artificial models of the computational behaviour of cells, tissues and ontogenetic systems gives weight to the notion that biological computation can be annealed to the artificial domain.

5.1 Implementation

The aim of the research presented in this paper is to investigate the concept of a wholly bio-inspired genetic programming system; one that abstracts from biology in both the variational and generative axis of GP behaviour. This is early work, and the system's application is limited to the relatively simple problem of combinatorial logic design; the evolution of digital, non-recurrent circuits. An overview of research applying GP to this problem can be found in [20]. Evaluation problems are shown in figure 3.

The experimental system is depicted in figure 4. The evolutionary framework, shown at the bottom, is a spatially-distributed cellular GA, where each cell in the population structure carries out an evolution strategy upon local state and inputs from surrounding cells. Cells are connected together by a network, the topology of which determines the processing behaviour of the population. A cell's evolution strategy, shown at the bottom-right, selects the fittest individual from the emigrants of those cells designated as inputs by the network. This immigrant then undergoes recombination with the local 'elite'; the fittest individual created so far within this cell. If the fittest child is fitter than the elite, then the elite is replaced with this child. The cell's emigrant, to which mutational noise may be applied to encourage local search, is the fittest individual out of the parents and the children. Note that if the immigrant is fit, yet sexually incompatible with the elite, it will pass straight through the cell without affecting local state. This encourages the preservation of a fitness differential within the population, increasing the scope for backtracking and hence preserving breadth of search. Retention of fit solutions through local elite, combined with deterministic selection, is thought to provide more structure to depth of search.

Figure 3: Arithmetic circuits used as test cases.



Candidate solutions to a problem are represented as shown in the top-left of figure 4. Solution genotypes compose a linear array of program elements. A program element consists of an activity, defining its role within the program, and a table of specificities, defining its affinity for outputs from other activities. An activity can be considered the combination of a function and a context, allowing a program to have multiple instances of a function with each having a different context. Hence, each instance can take on a different role within the program, and other activities can have specificity for the function within this context, rather than every instance of the function.

A specificity is a numerical value that defines a program element's relative affinity for output from a particular program element. Each data consuming element has a specificity for the output of every producing element. Consequently, the genotype can be visualised as describing a weighted network with vertices between every producer and every consumer; where the weight attached to a vertex indicates its likelihood of becoming a wire in the circuit. When a circuit is built from an enzyme program, specificities must be translated into wiring. A program element is ideally connected to the outputs of the two other elements that it has the highest specificities for. However, circuits are constrained so that no recurrent connections are allowed — which means that circuit elements will not always be connected to those for which they have highest affinity. Circuit output terminals choose their input gate according to specificity. Circuit elements downstream of the outputs then choose inputs for which, after any elements upstream have been removed from their choice, they have highest specificity.

Once this developmental process is concluded, the circuit is evaluated. This takes place through simulation, where the circuit's response to inputs is measured and a truth table is generated. Fitness is measured by comparison with the problem's correct truth table.

5.2 Results

Table 1 shows results for the one-bit adder problem. The population topology used for these experiments was a mesh with each cell connected to its four nearest neighbours (including top-bottom and side-side connections). The key parameter varied across experiments was the size of the dimensions of this mesh. Figure 5 depicts how this parameter affects the primary performance properties of the system — the number of generations required to find an optimal solution, and the proportion of runs which resulted in an optimal solution.

Generation count is measured using three metrics: mean, median and cumulative mean. Mean and median are raw measures, indicating average generation counts for success-

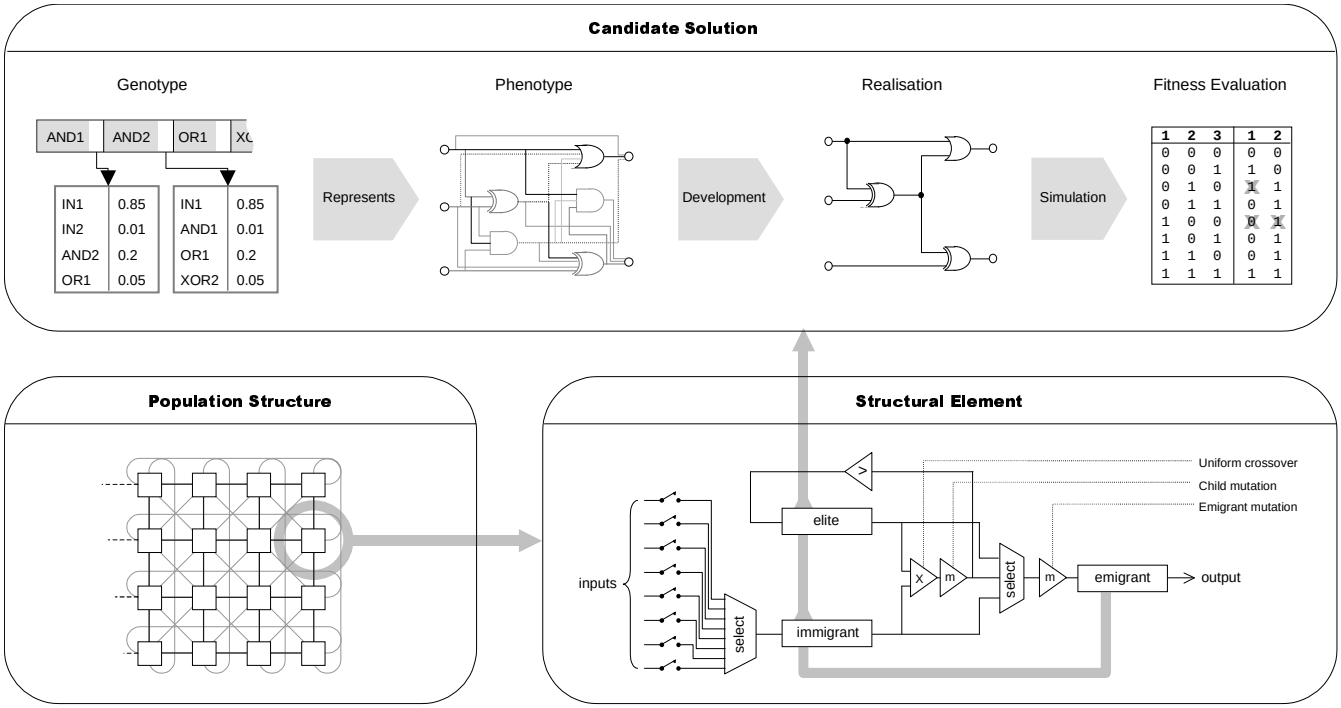


Figure 4: The Experimental System

ful runs over a number of trials. Cumulative mean takes into account the proportion of unsuccessful runs, giving a measure of the average expected wait between optimal solutions. Most runs return results within a fairly narrow range of values. A smaller number of results, especially in smaller populations, produce high generation counts. This is a reflection of the stochastic nature of evolutionary search and, whilst infrequent in large populations, is unavoidable. The worst of these runs are filtered by placing an upper limit on the generation count, in this case 200 generations. Runs which exceed this, and those that converge upon sub-optima, are classified as unsuccessful.

The functional profile shown in figure 5 generally agrees with the standard profile for a genetic algorithm based system. Increasing population size leads to an exponential decay in the generation count and an exponential increase in the success rate, both of which are observed to tail off at higher population sizes.

Table 2 shows some results for the two-bit multiplier problem. Optimal solutions are found on average about every 100 generations. Values of minimum computational effort, giving the number of solution evaluations required for a 99% certainty of success, have been calculated to allow a rough comparison with work by Miller [20]. For the adder and multiplier, respectively, they are 42,000 and 180,000. Miller cites results between 210,000 and 585,000 for the multiplier depending upon the formulation of the problem. Both approaches place different constraints upon the search space, so a direct comparison isn't yet possible. Nevertheless, these results do not appear any worse than Miller's.

5.3 Analysis

Figure 6 illustrates a typical run of evolution of a one-bit adder in a 12x12 population over a span of thirty generations, showing the correct solution to be a descendent of twelve first generation solutions. Fitnesses at key points are shown by measures of percentage-correctness.

This phylogenic trace illustrates certain behaviours apparent within the system. First, whilst mutation events are common within the run, only three of them map to pheno-

Table 1: Results for 1-Bit Adder

Population		Generations to optimum			Successful runs	Average suboptimum
Topology	Size	Mean	Median	C.Mean		
4x4	16	–	–	–	2%	87%
5x5	25	72	78	131	18%	92%
7x7	49	65	53	93	43%	94%
9x9	81	61	42	82	49%	94%
10x10	100	60	51	83	62%	94%
12x12	144	54	40	64	80%	94%
14x14	196	48	29	52	90%	94%
16x16	256	33	23	34	95%	94%
18x18	324	35	24	37	92%	94%
20x20	400	31	19	31	100%	–

Table 2: Results for 2-Bit Multiplier

Population		Generations		Successful runs	Average suboptimum
Topology	Size	Mean	C.Mean		
14x14	196	69	196	35%	95.6%
18x18	324	70	127	55%	96.4%
20x20	400	56	93	60%	96.2%

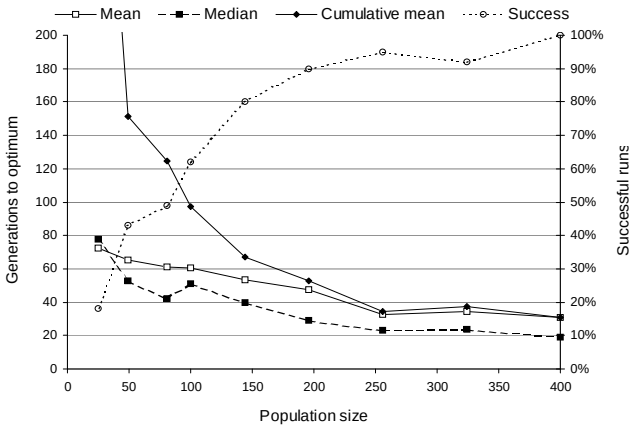
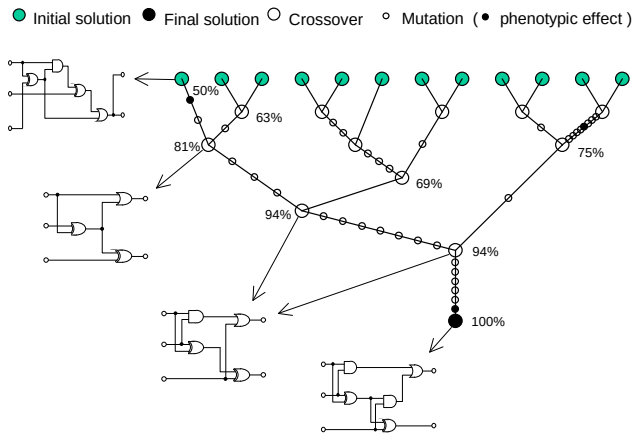


Figure 5: Algorithm Performance

typic changes. The remainder are synonymous mutations, silent mutations that either act upon untranscribed components of the genotype — components comparable to recessive genes — or act upon transcribed components in such a way that does not affect the nature of their transcription. Second, phenotypic changes are not always simple. In many cases, changes are compound; with several elements adapting in concert to achieve a high-level circuit transformation. Between the last crossover and the final solution (the lower two circuits), for instance, a single non-synonymous mutation introduces a new element into the circuit — an operation which requires, from a phenotypic viewpoint, entry of a new component in tandem with the modification of another component's input.

These two behaviours — synonymous and compound mutation — are termed, respectively, neutrality and macromutation. Both are a result of the separation of genotypic and phenotypic search spaces accorded by a genotype-phenotype mapping. Neutrality is a property which emerges from redundancy in this mapping. It is the ability of an individual to maintain phenotypic fitness invariance in the light of genotypic change (see [21] for a view on neutrality in digital circuit evolution). Significant movement in the genotypic search space without significant movement in the space of phenotypic fitness is called a neutral walk. Macromutation is the converse of neutrality; where a small movement in genotype space can lead to an apparently large movement in phenotype space. These behaviours allow greater expression during evolution than would be afforded with a representation where there is no mapping between evolutionary and executable forms.

Figure 6: Evolution of a 1-Bit Adder



Genotypes are an artifact of evolution; the memory of an evolutionary search process. Analysing the complexity of a biological genome, it is evident that the information contained is far more than is necessary for the construction and functioning of the organism. One interpretation, which sees the genome as a search point in evolution, is that the untranscribed portion is nothing more than garbage. However, since the genome is an artifact of evolution, and evolution is a pangenerational process, the genome is best seen not as a discrete entity but rather as a cross-sectional snapshot of evolution. Therefore we can imagine the genome as not only containing information relevant to the present in search, but also containing information relating to the *state* of search; information concerning the past and the future of search. Some of this past information may no longer be needed, and can therefore be considered garbage. However, past information can support backtracking in search whereas future information, which derives from the direction of, or potential for, change, supports depth of search.

The representation used in the enzyme GP system also maintains more information than is necessary to construct a single phenotype. This is particularly so in its current incarnation, where genotypes are fixed length and therefore forced to reference each component available to the system. In analogy to dominance hierarchies in biology, the genotypic elements that describe active components — those which are 'transcribed' to the circuit — are considered dominant; and those which remain hidden, recessive. There are two kind of component within the system: functional elements and connections; both of which can be dominant or recessive. Connections, or rather the specificities which describe connections, are the targets of mutation. Mutations targeted at recessive connections belonging to recessive elements will always be neutral. If the target is a recessive connection within an active component, the effect will most likely be neutral, though it could also cause the connection to become dominant over a currently active connection. This, in turn, might cause other components to become active. Mutations targeted at active connections in active components are most likely to reduce the strength of these connections (since mutation updates their values randomly, and high values are most likely to be replaced with lower values), which may then cause dominance to be lost to a recessive connection.

The result of changing expression due to changing dominance is particularly apparent when solutions are subjected to

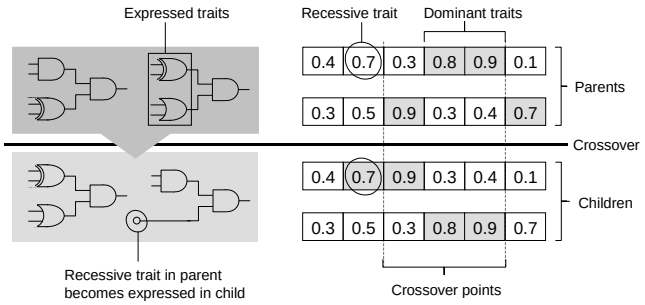


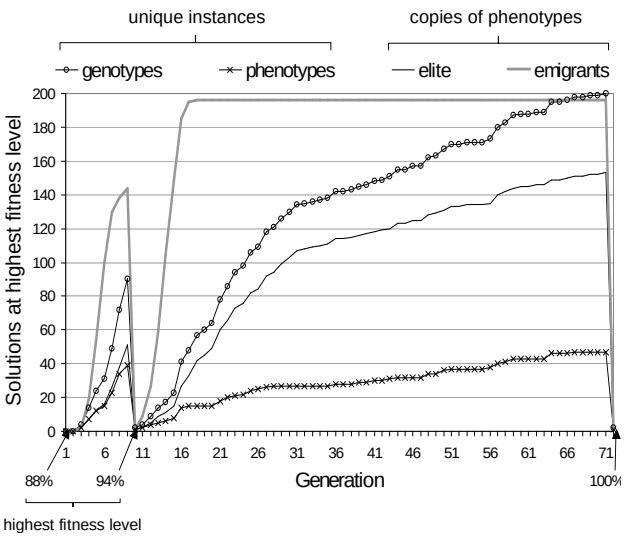
Figure 7: Effect of Crossover

crossover. To illustrate this, figure 7 shows one possible result of recombining a single AND gate between solutions. It is evident from the genotype of the first parent that the dominant connections are only marginally stronger than the strongest recessive connection, making this connection a feasible candidate for gaining dominance after crossover. In this example, this is exactly the result, and leads to a recessive trait becoming expressed in one of the child solutions.

From an evolutionary perspective, genotypes represent not one, but many phenotypes. Whilst only one phenotype is expressed in each generation, the choice of which phenotype is expressed is highly sensitive to the effect of mutation upon the precariously balanced dominance hierarchy of the genotype. The observable effect of this is macromutation; concerted or unexpected changes of the phenotype. It also makes individual genotypes efficient — since each one encodes many different solutions to the problem — meaning that a population has an information content more in line with a larger population of discrete solutions. However, by keeping related solution components together in a single genotype, it seems conceivable that evolution is able to make better use of this information than would be the case if it were dispersed throughout a population of discrete solutions. By this view, the untranscribed components of a genotype are the evolutionary state, or context, of the transcribed components — describing potential for change towards either previous solutions (backtrack points) or future solutions through changes in expression of transcribed and untranscribed components.

As well as encouraging diversity at the representation level, enzyme GP also preserves it at the population level. The fitness of a full adder is measured against only sixteen output bits and consequently has only three effective fitness brackets above the best solutions typically found in the seed population. Figure 8 plots the frequencies of solutions of the current highest fitness during a run of evolution: showing both the number of unique genotypes and phenotypes as well as their representation within the population's elite and emigrant slots. The graph shows that the genetic diversity is considerably higher than the phenotypic diversity; as would be expected given the level of neutrality in the representation. Fit phenotypes, broadcast from the cells where they were discovered, rapidly dominate the migrant population, spreading their wisdom in a manner akin to missionaries. The number of cells discovering, and therefore broadcasting, these fit solutions grows steadily in line with the generation of new genotypes through the recombination of lower-fitness elite and migrant solutions. The graph provides a good illustration of the high level of diversity accorded by both the genetic representation and the evolutionary algorithm. In the current system, only one optimal solution can be generated with the activities provided to the algorithm. However, it seems likely that if this constraint were lifted, then the algorithm could find multiple optimal solutions concurrently.

Figure 8: Solution Diversity



6 Conclusions

The hypothesis behind this research was that more effective evolutionary algorithms could be developed by paying closer attention to the details that make evolution effective within biological systems. The focus of the research reported in this paper was to test the assertion that the representation used for 'genetic programs' in biological organisms could be derived for use in genetic programming as a representation for other types of program. Whilst only a preliminary investigation of these concepts, it is hoped that the experimental results and analysis presented in this paper go some way towards proving these assertions.

7 Future Work

Immediate scope for future work includes a closer inspection of the dynamics and behaviours involved in the evolutionary process. Also, the system is currently constrained by user-imposed limits on the numbers and types of functional elements available to the system. An important step towards resolving this would be the allowance of variable-length solutions. In the long term, it is hoped that the system will form the basis for a more general approach to GP applicable to a

wider domain of problems; especially the evolution of computer software. However, a number of issues will have to be investigated before this is possible.

Complexity in biological systems emerges not only from the activities of metabolic pathways, but also from the configurational roles played by gene expression and signalling pathways. Genetic networks are systems of emergent biochemical pathways that, by controlling the format of metabolic pathways, serve as the predominant mechanism of control and coordination within organisms. Signalling pathways enable communication of data and control between separate metabolic pathways. Given the power that these configurational activities have in the organisation of biochemical systems, it would be interesting to investigate the effect of similar mechanisms within enzyme GP.

Bibliography

[1] T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, The University of New Mexico, 1995.

[2] G. P. Wagner and L. Altenberg. Complex adaptations and the evolution of evolvability. *Evolution*, 50(3):967–976, 1996.

[3] P. Angeline. Subtree crossover: building block engine or macromutation? In J. Koza et al, editor, *Genetic Programming 1997: Proceedings of the Second Annual Conference, GP97*, pages 240–248. Morgan Kaufmann, 1997.

[4] 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.

[5] R. Shipman, M. Shackleton, and I. Harvey. The use of neutral genotype-phenotype mappings for improved evolutionary search. *BT Technology Journal*, 18(4):103–111, October 2000.

[6] Lee Altenberg. The evolution of evolvability in genetic programming. In K. Kinnear, Jr, editor, *Advances in Genetic Programming*. MIT Press, 1994.

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

[8] 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.

[9] P. Cohen. High in protein. *New Scientist*, 168(2263):38–41, November 2000.

[10] P. C. Marijuán. Enzymes, artificial cells and the nature of biological information. *BioSystems*, 35:167–170, 1995.

[11] M. Conrad. Molecular computing: The lock-key paradigm. *IEEE Computer*, 25(11):11–20, November 1992.

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

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

[14] V. N. Reddy, M. L. Mavrovouniotis, and M. N. Liebman. Petri net representations in metabolic pathways. In L. Hunter, D. Searls, and J. Shavlik, editors, *Proceedings of the first international conference on intelligent systems for molecular biology*. AAAI, MIT Press, 1993.

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

[16] J.-L. Fernández-Villacañas-Martin, J. M. Fatah, and S. Amin. Computing with evolving proteins. In *Fourth European Conference on Artificial Life, ECAL97*, July 1997.

[17] H. Kargupta. Gene expression: The missing link in evolutionary computation. In D. Quagliarella, J. Periaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms in Engineering and Computer Science*, chapter 4. John Wiley & Sons Ltd, 1997.

[18] C. Ferreira. Gene expression programming: A new adaptive algorithm for solving problems. Unpublished, available via http from www.genetic-expression-programming.com, 2000.

[19] 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.

[20] 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.

[21] V. K. Vassilev and Julian F. Miller. The advantages of landscape neutrality in digital circuit evolution. In J.Miller, A.Thompson, P. Thomson, and T. C. Fogarty, editors, *Evolvable Systems: From Biology to Hardware (ICES2000)*, volume 1801 of *Lecture Notes in Computer Science*, pages 252–263. Springer, April 2000.