Chapter 1

THE CHALLENGE OF COMPLEXITY

Wolfgang Banzhaf
Department of Computer Science
University of Dortmund, Germany
banzhaf@cs.uni-dortmund.de

Julian Miller
School of Computer Science
The University of Birmingham, UK
j.miller@cs.bham.ac.uk

Abstract In this chapter we discuss the challenge provided by the problem of evolving large amounts of computer code via Genetic Programming. We argue that the problem is analogous to what Nature had to face when moving to multi-cellular life. We propose to look at developmental processes and there mechanisms to come up with solutions for this "challenge of complexity" in Genetic Programming.

Keywords: Genetic Programming, Evolutionary Algorithm, Complexity, Scaling Problem, Development, Heterochrony

Introduction

The purpose of this chapter is to pose a challenge to the sub-area of Evolutionary Computation (EC) dealing with algorithm evolution, Genetic Programming [20]. Genetic Programming (GP) has a fundamental mechanism which distinguishes it from other branches of EC, namely a means to adapt the complexity of its solutions [3]. Such a mechanism needs to be in place in GP since the resulting solutions are programs and algorithms, or, in other words, active entities which usually require input from somewhere that is subsequently transformed into output through the target program or algorithm.

It has been shown in recent years, that there are lower bounds on the complexity of solutions to algorithmic problems in GP [22]. Below a certain threshold, no algorithm would be able to perform a predefined task. Above that threshold, however, numerous programs would be able to perform the task. Evolution in GP is thus expected to lead the programs it breeds past this threshold, to be able (only after passing) to home in on one or the other of the many solutions that exist there. One might expect that GP would be well equipped to handle tasks of varying complexity because of its basic ability to adapt complexity.

As it turns out, however, GP is not able to handle complexity gracefully, it has a scaling problem. As is well known from other search algorithms, more complexity means larger search spaces. Larger search spaces in turn mean a combinatorial explosion in the number of possible solutions which need to be visited. Even a path-oriented algorithm like an evolutionary algorithm suffers from the problem of scaling under such circumstances. Although GP is regularly able to evolve programs of length 50 to 100 lines of code, this is a far cry from what would be needed to provide a useful method for day-to-day assistance for programmers.

Various remedies have been looked at over the years. Modularization of programs is one important method to improve scalability. The problem at hand is divided into sub-problems which are supposed to be less difficult (and thus would require less complex solutions). These subproblems could be solved in a divide-and-conquer method, whereby the overall solution is put together from the various sub-solutions evolved independently. Koza [21] has done an entire series of well thought-out experiments in order to show that GP is indeed able to proceed along those lines, provided it is equipped with appropriate means (ADFs in his approach). ADFs are good at structuring a global solution into parts, and by repeated use through calls from the main program with different arguments they provide reusability features for code in multiple subtasks. There have been other approaches toward modularization in the last decade [1, 4, 28], all trying to develop methods for better scalability.

However, all of these methods have failed to deliver on the fundamental challenge to GP which can be summarized in the following task:

Using GP, evolve a program whose purpose is so complex that it requires 100,000 or a million lines of hand-written code or 10,000 modules of average size 100 lines of code.

Application examples coming to mind are the following tasks

- Direction and control of the processes in a production plant

- Safe operation of an aircraft under a variety of weather conditions

- Design of a convenient multi-functional desktop computer tool, such as an editor or a mailer
- Maintaining a large network of computers as a self-repairing system
- Translation of one human language into another
- Recognition of pieces of art and music from visual or audio clues
- Evolution of a program playing Go with human-competitive performance
- A computer operating system based on self-regulation
- etc.

In other words, the challenge is to radically dispose of the complexity limits for the evolution of computer code, and aim at complexities heretofore only achieved by large teams of human programmers.

This chapter is therefore devoted to offering a possible solution to this challenge. This solution, however, can only be framed in very abstract, sometimes speculative words. Taken literally, it will not suffice to arrive at a workable mechanism. But the goal here is to set the mind of the reader into such a framework that she or he might come up with appropriate ideas to approach this challenge.

The rest of the chapter is organized as follows. Sec 1 summarizes very shortly the fundamental idea behind GP, Sec 2 looks at an ostensibly similar scaling problem situation in the area of Biology. Sec. 3 discusses Nature's way to deal with this problem, the introduction of a developmental process between the information storage in the genotype and the active entity, the phenotypic organism that results from its expression. Sec. 4 then tries to formulate a few principles of this solution to the problem that might be transferable into Genetic Programming. Sec. 5, finally goes one step further and proposes a possible scenario for the introduction of development into GP. Sec. 6 briefly discusses earlier experiences with the introduction of development, mostly treated under the heading genotype-phenotype-mapping.

1. GP Basics and State of the Art

Genetic Programming is part of the area of Evolutionary Algorithms which apply search principles analogous to those of natural evolution in a variety of different problem domains, notably parameter optimization. The major distinction between GP and these other areas of Evolutionary Algorithms is that GP controls active components like symbolic expressions or instructions as opposed to simple parameters, and that GP is

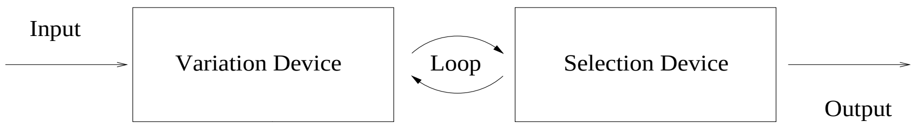

Figure 1.1. The variation selection loop of GP and other artificial evolutionary systems.

able to develop its own representation of a problem by allowing variable complexity of its individuals.

As other evolutionary algorithms GP follows Darwin's principle of differential natural selection. This principle states the following preconditions for evolution to occur via (natural) selection:

- A population of entities called individuals is formed which can reproduce or can be reproduced.
- There is heredity in reproduction, i.e. individuals produce similar offspring.
- In the course of reproduction variation occurs that affects the likelihood of survival and therefore of reproducibility of individuals.
- Due to excessive reproduction individuals are caused to compete for finite resources. Not all can survive the struggle for existence. Differential natural selection exerts pressure towards improved individuals.

Thus a variation and selection loop is iterated which constantly tries to improve solutions (see Figure 1.1).

The representation of programs, or generally structures, in GP has a strong influence on the behavior and efficiency of the resulting algorithm. As a consequence, many different approaches toward choosing representations have been adopted in GP. The resulting principles have been applied even to other problem domains such as design of electronic circuits or art and musical composition.

The mechanism behind GP works with a population of programs which are executed or interpreted in order to judge their behavior. Usually, a scoring operation called fitness measurement is applied to the outcome of the behavior. For instance, the deviation between the quantitative output of a program and its target value (defined through an error function) could be used to judge the behaviour of the program. This is straight-forward if the function of the target program can be clearly defined. Results may also be defined as side-effects of a program, such as consequences of the physical behavior of a robot controlled by a genetically developed program. Sometimes, an explicit fitness measure is missing, for instance in a game situation, and the results of the game (winning or loosing) are taken to be sufficient scoring for the program's strategy. The general approach is to test a variety of programs at the same problem and to compare their performance relative to each other.

The outcome of fitness measurement are used to select programs. There are a number of different methods for selection, both deterministic and stochastic. These selection schemes determines (i) which programs are allowed to survive (overproduction selection), and (ii) which programs are allowed to reproduce (mating selection). Once a set of programs has been selected for further reproduction, the following operators are applied:

- reproduction
- mutation
- crossover

Reproduction simply copies an individual, *mutation* varies the structure of an individual under control of a random number generator, and *crossover* mixes the structure of two (or more) programs to generate one or more new programs (see Figure 1.2). Additional variation operators are applied in different applications. Most of these contain problem-specific knowledge in the form of heuristic search recipes adapted to the problem domain.

In this way, fitness advantages of individual programs are exploited in a population to lead to better solutions. A key effort in Genetic Programming is the definition of the fitness measure. Sometimes the fitness measure has to be iteratively improved in order for the evolved solutions to actually perform the function they were intended for. The entire process can be seen in close analogy to breeding animals. The breeder has to select those individuals from the population which carry the targeted traits to a higher degree than others.

In the meantime, many different representations for GP were studied, among them generic data structures such as sequences of instructions or directed graphs, as well as more exotic data structures such as stacks or neural networks. Today, many different approaches are considered as GP, from the evolution of parse trees to the evolution of arbitrary structures. The overarching principle is to subject structures with variable complexity to forces of evolution by applying mutation, crossover and fitness-based selection. The results are not necessarily programs.

When analyzing search spaces of programs it was realized that their size is many orders of magnitude larger than search spaces of combina-

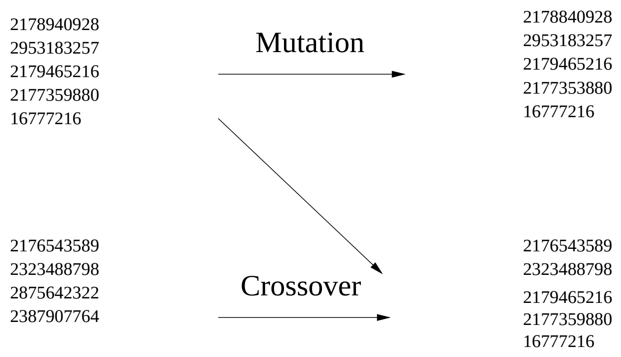

Figure 1.2. The primary operations of GP, mutation and crossover, as applied to programs represented by sequences of instructions. The instructions are coded as integer numbers.

torial optimization problems. A typical size for a program search space might be $10^{100,000}$, as opposed to a typical search space for a combinatorial optimization problem of the order of 10^{100}. Although this might be interpreted as discouraging for search mechanisms, it was also realized that the solution density in program spaces is, above a certain threshold, constant with changing complexity [23]. In other words, there are proportionally many more valid solutions in program spaces than in the spaces of combinatorial optimization problems.

2. The Situation in Biology

The situation in biology is also complicated. Life needs many supporting structures. Even single-cell organisms are already very sophisticated. Let's take the bacterium E.Coli as an example [16].

A bacterium is an autonomous living system and organizes molecules into a particular dynamic pattern that keeps it alive. Following Neidthardt et al, [27] there are a total of $300 \cdot 10^6$ molecules (excluding water with $40 \cdot 10^9$ molecules) in appr. 3250 different varieties (proteins, m-,t-RNA, DNA, lipids, small metabolites and ions, peptidoglycan, etc.). The genome of E.Coli is a single, circular molecule of $4.6 \cdot 10^6$ base pairs, which is to say it contains 6 Mbits of information (again accounting for reduncancy in the code) Notably, E. Coli has approximately 4300 protein coding genes (88 % of the genome) 0.8 % stable RNAs, 0.7 % repeats. 11 % of the genome might contain regulatory information (for a recent classification, see [31]). Mushegian and Koonin [26] identify a subset of 256 shared genes between two very simple bacterial organisms (H. influenzae and M. genitalium) which seem to provide the essential functions of life for those creatures. So how can all this multitude be organized by such a little genome?

Even more difficult is the situation in multicellular life. Take a human genome with its $3 \cdot 10^9$ nucleotides. Each nucleotide carries 2 bits, hence for a rough estimate we arrive at 4 Gbit maximum information content of the genome (the number was reduced from a simple multiplication, since due to code redundancy the information content is about 1/3 smaller).

On the other hand, take the number of cells of a human body as a rough estimate of the phenotype's information content: According to various estimates, the body amounts to approximately $50 \cdot 10^{12}$ cells. The estimate is raw and difficult to quantify more accurately because the number of cells changes dynamically. Cells are produced and die during the life of an individual. Now assuming that each cell has an information content of at least 1 Mbit, this results in the requirement for $5 \cdot 10^{17}$ bits $> 10^8 \cdot 4$ Gbit, or 10^8 times the human genome! Note that the estimate of 1 Mbit per cell is unrealistically low, as we shall see when we consider free-living single-cell creatures. According to Calow (1976) [7], the cells of the human body have to be weighed in with a much larger information content, resulting in a total of $5 \cdot 10^{28}$ bits for the body!

We can see easily, that these numbers are completely out of proportion, which means that the information in the genome must be used in a sophisticated way so as to produce a viable organism. We might call this the information dilemma of the genotype-phenotype relation.

3. Nature's way to deal with complexity

In his now famous book 'The way of the cell' biologist Frank M. Harold explains: "Genes specify the cell's building blocks; they supply raw materials, help regulate their availability and grant the cell independence of its environment. But the higher levels of order, form and function are not spelled out in the genome. They arise by the collective self-organization of genetically determined elements, effected by cellular mechanisms that remain poorly understood." [16], p.69.

Thus, there are other aspects of natural biochemical systems, so far not fully understood, that structure interactions and determine the fate of molecules. These aspects constrain the possible directions that genes could affect their products. Self-organization and self-assembly are among them as are physical (and other) laws. In addition, the natural abundance of certain materials, energy, or even information plays an important role. These aspects are providing the environment in which a living system is supposed to survive.

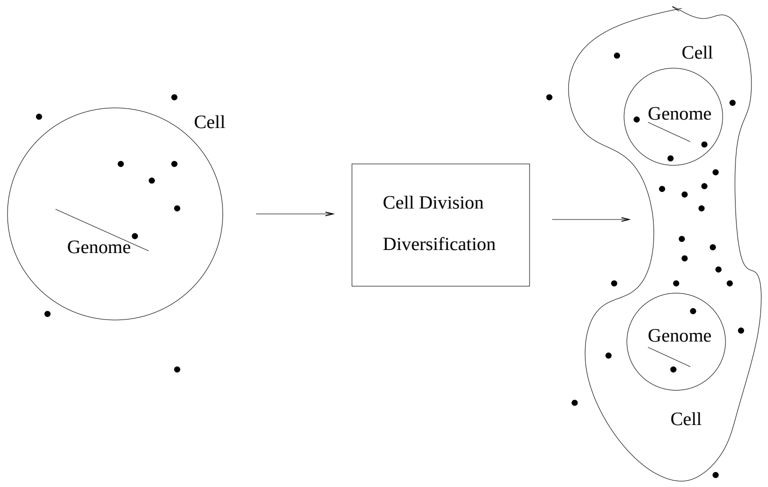

Figure 1.3. Single cell and multi-cellular system. The environment of a genome is primarily the cell in which it is residing. Control is exerted both by the cell and its environment via substances (black dots) diffusing around in intra- and extracellular space. The genome in turn tries to influence its environment by providing orders to produce certain substances. If a multi-cellular being is constructed a division and differentiation process is set into motion which leads to a number of cells with a boundary to the outside environment. The organism is the primary environment of a cell, with intra- and extra- organismal message transfer via molecules (black dots).

Nature's self-organizing properties are beginning to be seen in all scientific and technical disciplines [5]. But is self-assembly without a genome sufficient to explain the intricate organization of a cell? For example, if all the necessary substructures and molecules were present in a medium, would they be able to form an E.Coli bacterium? Here we follow again the argument of Harold [16] and Rosen [29]. The answer is "No", because self-assembly can never be a fully autonomous process. In addition, some cell components cannot be formed by self-assembly since they need to be formed by, e.g., cutting and splicing. Further, membrane proteins catalyze directional reactions (uni-directional through the membrane) [17]. The direction itself is, however, provided by the cell, not by the amino acid sequence of the protein or its gene. More generally, a great deal of localized behavior takes place within a cell. Localization, however, cannot be provided by the genes, it is a feature of their environment, i.e. of the cell (see Figure 1.3).

The conclusion is inevitable: Cells do not self-assemble. But how do they succeed instead? They grow! Rudolf Virchow (1858) was the first to formulate this realization[1] in a now famous biological law: Omnis cellula e cellula (every cell originates from a cell). No cell has not come from another cell.

In other words, there is a tight coupling between what the genome instructs and what natural laws and resources in the environment allow the cell to do. In a way the genome exploits all physical laws available (together with all sorts of material, energy and information fluxes) in order to organize a living being.

The real trick of Nature was to hit upon a system of organizing characters (RNA, then DNA and protein) that allows open-ended evolution to proceed. That is to say that the system does not close down upon encountering enormous complexity, both in the environment and in handling its inner mechanisms. Clearly, only a combinatorial system has enough power to grow to each level of complexity demanded (and also to shrink to a lower level if necessity dictates).

"Biological forms are not fragile or contrived, quite the contrary, they are the 'generic forms' most likely to be found by self-organizing dynamic systems, and therefore both probable and robust. We may imagine systems 'exploring the space' available to the particular dynamics of each kind, and see evolution as the process by which their morphologies are transformed one into the other." [16], p. 198. It may be added that natural evolution is an opportunistic process in the sense that whatever works is exploited as much as possible. Thus, the notion of a very limited exploration of the design space, as put forward by Gould [12, 13] can be brought into agreement with the above opinion.

Going back to the question of how development could organize the massive amount of molecules into orchestrated multicellular organisms, it seems to us that the exploitation of the natural (physical) tendencies to self-organize, i.e. to form self-maintaining networks of structures on which matter, energy and information flows, is the key recipe that genomes use. In other words, genomes are specifying or, better, influencing the interactions that lead to these networks and take place in them. What was built on top of single-cell life, then, were elaborate mechanisms for cell communication and differentiation, based on the same principles as single-celled life was. The enormous number of genes added to single cell organisms can be put to use for the purpose of (a) adaptation of the cells to multicellular environments and (b) coordination between cells, a task that is obviously very complicated.

[1]One should be careful to include both (i) scaling up and (ii) diversification / specialization in one's notion of growth.

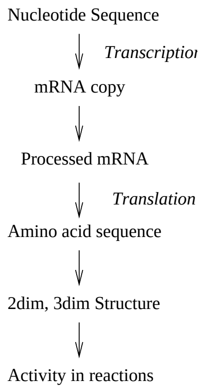

Figure 1.4. Transcription and translation as two important steps in the process of mapping information from genotype to phenotype.

A proper definition of biological development is in order here. At present, biological understanding might be summarized in the following statement: Development is a differential transcription (and translation) of genes in different cells and tissues at different times and rates, with each step ultimately initiated by the transcription and translation of the previous step.

The operations of transcription and translation probably warrant some explanation. Figure 1.4 shows the typical sequence of events from DNA to protein activity. After the mRNA copy is transcribed from DNA, it is processed and transported out of the nucleus of the cell. It then is translated at a ribosome into a sequence of amino acids which fold into a native structure able to perform biochemical activity.

The control and timing of transcription and translation in cells is called regulation and can be imagined as follows: The products of certain genes are not used in building the organism directly but rather are used to interact with other genes' products, with environmental cues, or with the DNA of other genes (both expressed and non-expressed parts thereof). By interaction they change the course of events in a cell, depending on the presence of interaction partners and the strength of their mutual effects. In this way, networks of interaction are formed among genes, called regulatory networks. As already mentioned, however, genes do not restrict their interactions to other genes, but may also interact with environmental material. In this way, they can interfere with another network of biochemical reactions that is formed within a cell, the metabolic network. However: "Genes seem to be distant from the biochemical network, maintaining control only by carefully timed "injections" of their products into crucial "branching points" where small inputs have big effects." [16]. Notably, most of the order of a cell is created by the underlying network, with only occasional but decisive intervention by genes.

In summary: The biochemical network of interactions between substances is the underlying substrate of a system of control built upon the effect of additional substances (signalling substances), that are itself produced by genes. The system is highly combinatorial in that many of the biochemical (maintenance) substances can interact with each other and with the signal substances. It is through a control of the expression of the where and when of the signals that genes exert their control on the underlying networks.

One other key insight of developmental biology is the notion of heterochrony [15, 11, 24] which seems to be able to explain a whole plethora of phenomena found in the developmental process (and in evolution, for that matter) [25]. Heterochrony describes the fact that during differentiation, a large amount of control can be exerted on development by controlling three variables only: (a) the onset, (b) the rate and (c) the offset of the expression of certain genes. As such, the phenomenon is not very much different from what must happen in single-celled organisms where, in response to changing growth and environmental conditions, certain genes alter their rate of expression.

Though we don't have much space to delve into this very interesting phenomenon, one angle on heterochrony is worth looking at more closely: Its relation to the discovery of novelty in evolution. Citing McKinney and McNamara, [24]: "Heterochrony is the cause of most developmental variation and heterochrony can cause major novelties. The main reason for heterochrony to be able to cause major novelties, even new tissues is the fact that it can alter the regulative development of cells already early on in development which will give rise to major "jumps" in morphospace." And later, the authors write: "Heterochrony can be applied at different levels (molecular, cellular, tissue, organism). It is interesting to note that "small rate or timing changes at the lower levels will often translate into complex result at the higher levels. The nonlinearity of the system will amplify some changes (pos. feedback) and dampen others (neg. feedback) as they cascade upwards." [24], p.48.

May it suffice to add one more key insight of developmental biology that is just starting to surface in detailed studies of early embryonic development of multicellular organisms: Interesting recent results suggest that the control of timing of developmental events, i.e. the actual mechanism by which heterochrony can be enacted, is due to an encoding of time and strength of expression of genes into the strength of interaction between (regulatory) genes [2, 9, 10].

4. What we can learn from Nature?

In the previous discussion we have seen some similarity to the problems in Genetic Programming. So a natural question would be what we could learn from Nature. Here we list a few of the aspects of the developmental process in Nature which might provide hints to our efforts in artificial evolutionary systems.

1 Nature stands before what we have called the information dilemma: How to instruct a body with so few genes? The size of a genome is very small for to provide the required information for a phenotypic organism. Nature's recipes are:

- The channeling or canalizing of environmental complexity (information, energy, matter, laws, interactions, dynamics, boundaries) into the developing phenotype. The complexity of the organism stems mainly from outside and has not to be provided by the genotype. The genotype mainly directs the assembly.
- The stability of an organism (whether mature or developing) is a steady state, not a static equilibrium. It is in a continual state of growing and dying to maintain itself. Nature is dealing with open systems (due to physical constraints) where energy and entropy considerations are important. Responsiveness to environment is much better this way.
- Development allows for open-ended evolution since it is a constructive process where layers of complexity are built onto each other (with the possibility of ever larger complexity).
- There is a built-in tendency of development to be recursive (see L-systems), which allows hierarchy-building in a very natural way.
- Development happens by way of communication between cells, i.e. it's a social system of cells. More generally, there are many combinatorial subsystems interacting with each other, erecting networks of communication flow.
- Fitness tests for phenotypic organisms are always punctual, i.e. individuals are never tested completely and therefore considered ready. Instead, multi-functionality is important and punctual fitness tests which would test for, e.g., metabolism efficiency today and for, e.g., adaptive capabilities tomorrow, allow for it to develop.

2 Time is the most important aspect of development. It results in the formation of a 4D space in biological development.

- Time and dynamics is a key to survival in real-time environments. No wonder it plays the major role in development also.
- Different time-scales (usually required by the environment) are easy to achieve, since development is intrinsically hierarchical.
- The time dimension is a way to "mold" results of development, as can be seen by the notion of heterochrony.
- There is labor division (and gradually more so) in the course of development.
- Incremental fitness is an important concept, too, i.e. there is a requirement of primitive functionality from the very beginning which is gradually refined until the organism is "mature".
- In terms of fitness landscapes: The fitness landscape gradually sharpens (becomes more rugged) in the course of development.
- The develomental process has an enormous degree of fault tolerance. Repair mechanisms are abound, as well as adaptability, and the ability of regeneration.
- There is a chain of being - from the first living thing to the last cell in a multi- cellular individual. This would be interrupted without development.
- Sexuality requires a 1-cell stage for each living being (for the uniqueness of information exchange in recombination). Thus Nature needs a mechanism for an organized transition from the one-cell stage to the multi-cell individual.

3 The mechanisms of development are constructive

- Starting from a single cell, whole bodies are constructed, consisting of millions and billions of cells.
- Development erects networks (metabolic, signaling, regulatory) of increasing complexity, within and between cells.
- Development makes use of neutrality, i.e. there are some phases in development where nothing happens if looked at from the behaviour of the phenotype.

- Development allows the exploitation of side-effects, perhaps in a very efficient way. Side-effects are an important source of innovation for evolution, since they are unintentional effects which turn out to be useful for other purposes. Producing side-effects is what development can do, discerning their usefulness is left to evolution.

5. A possible scenario: Transfer into Genetic Programming

A linear genetic program is a sequence of instructions that is followed one by one. This might be a good way to organize a genome, as the subsequent execution of steps is a rather natural way of following this information. However, it is not a very natural way to look at program behavior, i.e. the phenotypes. We propose that, instead, complex programs of the type of interest here should be considered as networks of interacting objects which are to behave in complicated ways depending on the flow of input and required output. Thus, if one were to set up a system of interacting objects, designating input and output objects and their communication means, one would have a natural analogue to a biochemical network. Note that this does not necessarily imply that we ought to consider non-sequential programs here. Rather, it is the more general case.

Figure 1.5 shows a dataflow graph of a program phenotype. This is the graphical translation of the following program (line with "!" are not contributing to fitness):

```
void gp(r)
  double r[4];
{
   ...

   r[3] = r[1] - 3;
   r[1] = r[2] * r[1];
!  r[3] = r[1] / r[0];
   r[0] = r[1] - 1;
   r[1] = r[2] * r[0];
   r[1] = r[0] * r[1];
!  r[0] = r[2] + r[2];
   r[2] = pow(r[1], r[0]);
!  r[2] = r[0] + r[3];
!  r[0] = r[3] - 1;
!  r[1] = r[2] - r[0];
!  r[3] = pow(r[0], 2);
!  r[2] = r[2] + r[1];
   r[0] = r[1] + 9;
   r[0] = r[1] / r[3];
```

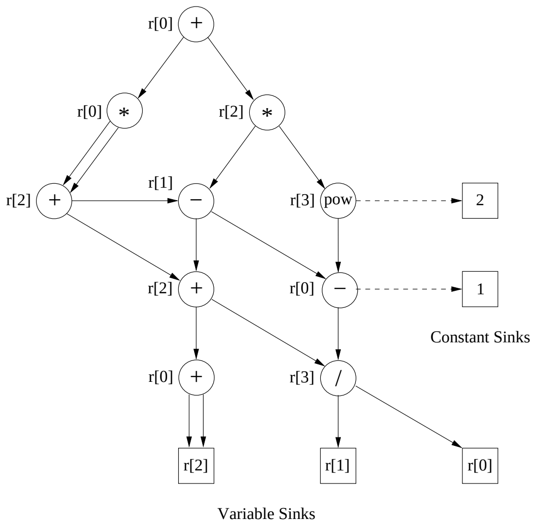

Figure 1.5. The network of data flow on registers as one example of program phenotype. The corresponding program is listed in the text as a linear sequence of instructions. Adopted from [6]

```
!  r[0] = r[2] * r[2];
!  r[2] = r[1] * r[3];
!  r[0] = r[0] + r[2];
}
```

In the language of object-oriented programming, objects possess attributes for receiving messages and methods for sending messages and performing other functions. Input driving the program system would be considered an information flow to be taken advantage of for a given purpose (output), and the right combination of interactions would be searched for by a genetic search method on the level of genes. Taking advantage would mean that those networks are differentially selected that perform, after development, the prescribed task better than others.

As for the "reconfiguration" of the object network, this would not happen through a direct modification of objects but rather through additional layers of message producing objects, and through their corresponding messages. The messages would act like signaling substances and interfere with the object network in a constructive way, e.g. by inhibition or by excitation, and the objects producing these messages would be genes located in sequence on a genome. Thus each gene would specify an object, where such an object would be even allowed to interact with other objects' specification of products. The most difficult part is presumably the latter, since it requires the ability for self-modification.

Perhaps one could even go down to the level of instructions (as equivalent to objects in the above sense). Instructions have an operation (through the op-code) and operands (input and output) to digest messages. The problem with instructions is that the desired behavior of a program needs to come about by side-effects of instructions only. Since there are flags to be set by instructions, in principle there could be a way. It would have to be decided, what the side effects are (1. one could select for the flags, or 2. one could select for values in registers and define a network of interactions between flags of instructions, although that might be more difficult).

How could heterochrony come into play? The idea would be to influence the underlying information flow in the network of objects by means of variation in "timing" of expression. That would be a very smooth way of variation, even expressible directly as a simple parameter evolution. On the other hand, Nature's example teaches us how to translate timing signals into pattern matching. So there would be another way to control time-dependent development.

Finally, the network of interacting objects could be built up by a developmental process, perhaps starting from one object. In this case the object would act like a cell with its genome directing the expansion into a larger network of interacting objects, possibly using pre-defined objects that have been specified already and only need to be coopted into the network.

Perhaps we have ventured too far now. However, we know that a simple division and diversification process of objects can reach any size of a network in logarithmic time. As such it is perfectly imaginable that the process envisioned here will quickly reach the desired complexity for any prescribed task. Nevertheless we have to leave it to the reader and further considerations how such a scenario could be realized in a computer.

6. Conclusion

Some ideas related to the present contribution have been published in the past. Notable is Gruau's [14] system of cellular encoding which uses a grammar tree to produce programs in a simple developmental process for GP. This work has later been applied by Koza [21] and others to produce electric circuit designs. Cangelosi [8] was the first to try to make use of heterochrony in the context of GAs. A number of people are working on regulation and evolutionary algorithms using regulation, like in the work of Kennedy et al. [19]. The genotype-phenotype mapping has been studied in different papers, see for example [30] and just recently has been the subject of a special journal issue [18] under the heading "gene expression computing".

In the present contribution we have tried to provide a challenge to Genetic Programming which would be worth to meet in the long run. We have argued that Nature had to solve an analogous problem which it did by inventing the developmental process. We have discussed a number of aspects of development that seemed to us relevant in the context of artificial evolutionary processes, and sketched one way to achieve a similar mechanism in GP. It remains to be seen whether GP can meet that challenge in the future.

Acknowledgments

W.B. acknowledges the hospitality of the the Institute of Genomics and Bioinformatics at UC Irvine under its director Pierre Baldi, where part of this work has been written.

References

[1] Angeline, P., Pollack, J. (1994) *Coevolving high-level representations*, in: Proc. Artificial Life III, C. Langton (Ed.), Addison Wesley, Reading, MA 55 - 71

[2] Arnone, M. (2002) *Bringing Order to Organogenesis*, Nature Genetics, 30, 348 - 350

[3] Banzhaf, W., Nordin, P., Keller, R., Francone, F. (1998) *Genetic Programming - An Introduction*, Morgan Kaufmann, San Francisco, CA

[4] Banzhaf, W., Banscherus, D., Dittrich, P., (1999) *Hierarchical Genetic Programming using local modules*, Tech. Report Nr. CI-56/99 of SFB 531, University of Dortmund

[5] Banzhaf, W. (2002) *Self-Organizing Systems*, Encyclopedia of Physical Science and Technology, Academic Press, New York, Vol. 14, 589 - 598

[6] Brameier, M., *Linear Genetic Programming*, PhD thesis, Department of Computer Science, University of Dortmund, 2003, to appear

[7] Calow, P. (1976) *Biological Machines: A cybernetic approach to life*, E.Arnold, London

[8] Cangelosi, A., *Heterochrony and adaptation in developing neural networks* In W. Banzhaf et al. (Eds), Proceedings of GECCO99 Genetic and Evolutionary Computation Conference. San Francisco, CA: Morgan Kaufmann, 1241-1248.

[9] Davidson, E.H. (2001) *Genomic Regulatory Systems*, Academic Press, San Diego

[10] Gaudet, J., Mango, S.E. (2002) *Regulation of Organogenesis by the Caenorhabditis elegans FoxA Protein PHA-4*, Science, 295, 821 - 825

[11] Gould, S.J. (1977) *Ontogeny and Phylogeny*, Belknap Press of Harvard University Press, Cambridge, MA

[12] Gould, S.J. (1980) *The Evolutionary Biology of Constraint*, Daedalus 109, 39 - 52

[13] Gould, S.J. (2002) *The Structure of Evolutionary Theory*, Belknap Press of Harvard University Press, Cambridge, MA

[14] Gruau, F., *Genetic Synthesis of Modular Neural Networks*, in: S. Forrest (ed.), Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93, 1993, Morgan Kaufmann, San Francisco, 318–325

[15] Haeckel, E. (1866) *Generelle Morphologie der Organismen*, Reimer, Berlin

[16] Harold, F. (2001) *The Way of the Cell*, Oxford University Press, Oxford

[17] Harold, F. (2001) *Gleanings of a chemiosmotic eye*, Bioessays 21, 848-855

[18] Kargupta, H., *Editorial: Computation in Gene Expression*, Genetic Programming and Evolvable Machines, 3 (2002), 111 - 112

[19] Kennedy P.J., Osborn, T.R., *A Model of Gene Expression and Regulation in an Artificial Cellular Organism*, Complex Systems 13, 2001

[20] Koza, J. (1992) *Genetic Programming*, MIT Press, Cambridge, MA

[21] Koza, J. (1994) *Genetic Programming II*, MIT Press, Cambridge, MA

[22] Langdon, W. (1999) *Scaling of Program Tree Fitness Spaces*, Evol.Comp. 7, 399 - 428

[23] Langdon W 1999 Boolean function fitness spaces. In: Poli R, Nordin P Langdon, W and Fogarty T (eds.) *Proceedings EuroGP'99.* Springer, Berlin

[24] McKinney, M., McNamara, K. (1991) *Heterochrony: The Evolution of Ontogeny*, Plenum Press, New York

[25] McKinney, M. (1999) *Heterochrony: Beyond words*, Paleobiology 25, 149 - 153

[26] Mushegian, A., Koonin, E. (1996) *A minimal gene set for cellular life derived by comparison of complete bacterial genomes*, Proc. Natl. Acad. Sci. (USA), 93, 10268 - 73

[27] Neidhardt, F.C. (1996) *Escherichia Coli and SOlmonella typhimurium*, ASM Press, Washington, DC

[28] Rosca, J., Ballard, D. (1994) *Hierarchical selforganization in genetic programming*, in: Proc. of the 11th Int. Conf. on Machine Learning, Morgan Kaufmann, San Mateo, CA, 252 - 258

[29] Rosen, R. (1994) *Life Itself*, Columbia University Press, New York

[30] Smith, T., Husbands, P., O'Shea, M., *Neutral Networks and Evolvability with Complex Genotype-Phenotype mapping*, in E. Kemelen and S. Socik (Eds.), Proc. 6th ECAL-01, Prague, 2001, Springer, Berlin, 2001, 272 - 281

[31] Thomas, G.H. (1999) *Completing the E. coli proteome: a database of gene products characterised since completion of the genome sequence.* Bioinformatics 7, 860 - 861