

Laws and Life

Peter Grogono

GuoRong Chen JunFeng Song

Tao Yang Lei Zhao

Department of Computer Science

Concordia University

Montréal, Québec, Canada

grogono@cs.concordia.ca

ABSTRACT

Evolutionary Programming, an area of research in which software development exploits analogies with biological evolution, is already popular and is gaining respectability. An examination of the basic ideas underlying this research reveals incorrect assumptions and restricted approaches that limit the potential value of the results. We identify four issues that we believe to be important in the foundations of evolutionary programming and we describe a project designed to investigate their implications. We conclude with a discussion of possible applications of the research.

KEY WORDS

Evolutionary programming, genetic algorithms, emergence, evolution

1 Introduction

The whole real guts of evolution—which is, how do you come to have horses and tigers, and things—is outside the mathematical theory. C.H. Waddington [16]

Throughout history, people have looked for analogies between biology and technology. Recently, interest has focused on analogies between Darwinian evolution and software. Interdisciplinary researchers have asked two questions. Can software simulations provide insight into biological mechanisms? Can ideas from biology contribute to advances in software technology? Our work focuses on the second question: we believe that insights obtained from biology can help us to develop new ways to program computers.

The idea that evolutionary concepts can help to solve programming problems is about forty years old. John Holland designed the first evolutionary program in 1962 [8]. The field now has a name—Evolutionary Programming (EP)—and widely used techniques have been developed. The field has expanded and, today, evolutionary programming is but one branch of a wider discipline that has come to be known as ‘Artificial Life’

(AL). We have identified four critical issues in AL and we introduce them in Section 2.

Evolutionary Programming (EP) has many forms. We use the term inclusively so that, for example, we define Genetic Algorithms (GA) as one kind of EP. Most of these forms share a few key features: there is a *population* of individuals; individuals *reproduce* by creating new individuals similar to themselves; an individual has a *fitness* that is used as a criterion for *selection*; and, as the population evolves, the average fitness is expected to increase. In many, although not all, cases, the behaviour of an individual is partly or wholly determined by a descriptor called its *genome*.

We have initiated a research project, described in Section 3, to explore the issues. The project is built around a simulation program that we discuss in Section 4. In Sections 5 and 6, we discuss related work and applications, respectively, and in Section 7 we present our conclusions.

2 Research Issues

2.1 Optimization

EP is well suited to optimization and EP research is dominated by the study of optimization problems. Although effective optimization algorithms have many practical uses, the emphasis on optimization has inhibited research in other directions that might be more fruitful in the long term.

Although evolution in nature may lead to optimal solutions in some cases, optimality is not the usual result of evolutionary development. Evolution produces solutions that are *good enough*—they enable the majority of individuals to survive—and, more importantly, evolution maintains a *family of related solutions* rather than a unique, optimal solution. In biology, the family of solutions is embodied in the ‘gene pool’ of a species. This is the major difference between genetic optimization programs and biological systems. There are also a number of differences between the way ‘genes’ are typically used in EP and in biology that are significant for future research.

- ◇ In biological life, all but the simplest organisms (phages) contain reproductive machinery to build copies of themselves. This distinguishes them from most work in EP, in which replication is performed by ‘magic’ (that is, by the controlling software). Tom Ray’s **tierra** [14] and Adami’s **avida** [1] are exceptions to the general rule.
- ◇ A biological phenotype is not fully determined by its genotype; the form of a newborn organism depends on the reproductive environment as well as on the genome. This implies that the genome is not a complete and self-sufficient description of the organism. You can’t make a dinosaur from dinosaur genes: you need a female dinosaur, too.
- ◇ In biological life, the structure of the genome and the genetic code itself can evolve [15, 17].
- ◇ In biological life, the genome of an organism affects the entire life of the organism, not just its form at birth. Genes, for example, control aging and genetic diseases may appear for the first time late in life. Each cell uses its copy of the genome continuously, enabling the genes that synthesize the proteins that it needs and disabling others.
- ◇ In biological life, most genes do not code for a single characteristic. Changing a single gene may have several effects, and the effects may be good or bad for the organism’s survival.
- ◇ In biological life, the genome plays a dual role. It is treated as *data* when it is copied from parent to child. It is treated as *instruction* when it is used to build and maintain the organism.

#1 *Limiting EP to the solution of optimization problems is both non-biological and unnecessarily restrictive.*

2.2 The Search Space

It is very easy to be impressed by simple evolutionary programs. Suppose, for example, that you are shown a program that simulates the evolution of fish. The demonstration begins with clumsy spherical objects that gradually develop fins and tails, become streamlined, and turn into fish-like organisms. This appears impressive, but it is quite possible that the program is simply choosing parameters that determine the shape of the body and fins of the fish. The hard part of the problem, which is the decision to use fins for swimming, has been made by the programmer; the program is simply choosing the best size, shape, and position for each fin. The fish simulation must be judged in terms of its search space: the shape of fins. A simulation will not create fish with propellers because fish are defined as being finned creatures.

#2 *It is important to distinguish the search space accessible to the simulated organisms and the potentially much larger search space for the problem.*

The issue of search space is clearly related to the issue of optimization. Our goal is to use EP for innovation, which implies a very large search space, rather than for optimization, for which a smaller search space is sufficient.

2.3 Terminology

When we move outside the domain of simple optimization problems, we find that claims depend on a kind of anthropomorphism. In this area, anthropomorphism takes the form of interpreting crude mechanical behaviour as though it were biological. For example, simulated agents are said to ‘eat’, ‘pursue’, ‘flee’, ‘fight’, ‘reproduce’, and so on. The activity described as ‘eating’, however, bears no relation to a complex physiological process: it may mean merely that one number, referred to as ‘food in the environment’ is decreased, and another number, referred to as ‘food in the stomach’, is increased. The danger of anthropomorphism is that we may be misled into believing that we are actually simulating the named activity, with the result that we read more into a computer simulation than is actually happening.

#3 *Resorting to biological language that does not accurately represent the simulated behaviour is misleading.*

2.4 Complexity

Stephen Jay Gould argued (e.g., in [7]) that evolution does not take the form of a monotonic increase in complexity from amoeba to plant to ape to Charles Darwin. The tree of life is not a spindly Victorian tree of progress but a broad bush in which most branches lead to extinction. Gould did argue, however, that evolving organisms will tend to increase in complexity as time passes. The essence of his argument is that evolution leads to diversity: along any dimension, the mean value of a population may or may not change, but the standard deviation will increase. If the dimension is bounded asymmetrically, however, the mean will tend to move away from the boundary. In Figure 1, the vertical bar at the left represents the lower bound for complexity of living organisms. Since complexity has a lower bound—an organism may be too simple to survive—the mean complexity of organisms will tend to increase.

Stephen Wolfram does not share Gould’s view. He argues that complexity arises from very simple mechanisms [18]. He shows, for example, that a one-dimensional cellular automaton can simulate a Turing

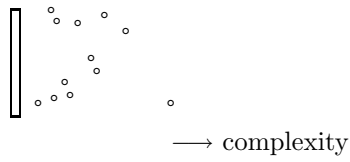


Figure 1. Evolving complexity

Machine—in other words, it can perform any computation in the currently accepted sense of ‘computation’. (We note, however, that although the *rules* of the automaton are simple, the computation may require unbounded time and space.) Wolfram argues that evolution is a *simplifying* mechanism: if we start with a collection of simple automata with rich behaviour, evolution will find the simplest and most economical mechanism that suffices to solve the problem. Although Wolfram presents many interesting patterns, he does not establish any definite connections between cellular automata and our actual world.

Neither argument seems to explain the phenomena adequately. We conjecture that both evolution and the emergence of complex behaviour from simple systems are critical for understanding life.

#4 *The arguments of Gould and Wolfram are not contradictory but complementary—evolution can both discover automata with rich behaviour **and** select those that work in the simplest possible way.*

3 The FormAL Project

The main focus of the FormAL project is the development of a program, described in Section 4, that we can use as a ‘sandbox’ to explore the issues we outlined in the Section 2.

The first issue that we address is anthropomorphism. Our treatment of this issue explains the name of our project and of this paper. The ‘laws’ and ‘life’ of the title are analogous to logic and artificial life.

The original goal of logic was to provide a precise foundation for thought. Initiated by Aristotle [2], logic was formalized in the nineteenth century by George Boole [3]. Logicians distinguish between a *theory*, which is purely symbolic, and a *model*, which is consistent with the theory but is associated with the world in some way. By analogy, we believe that it is important to distinguish between a *theory*, which consists of precise rules and a *model*, which associates the theory to biological phenomena. We call this approach ‘formal artificial life’ or ‘FormAL’ for short.

To address the first issue, the limitations of optimization, we decided to design a problem that cannot be reduced to mere optimization. This is easier said than done. A model that uses evolutionary techniques

must have a population of individuals that are selected according to fitness, as described in Section 2. Since the fittest individuals survive, the result of simulating the evolution of a population is to maximize fitness—thereby solving an optimization problem!

The way out of this trap is to recognize that ‘fitness’ is not a biological phenomenon but is rather an artifact imposed on biological systems by theorists. Optimization is not entirely absent from nature, but an optimized trait appears only in particular circumstances. For example, our eyes are ‘optimal’ in the sense that if they were any more sensitive we would ‘see’ flashes of light caused by quantum fluctuations. Nevertheless, a cat has better night vision and some birds have more acute vision than we do. Our eyes represent a compromise between vision that is affordable and vision that is optimal. That compromise defines biological fitness for a particular organism.

In our simulation, individuals, which we call *agents*, must solve the problem of surviving in a complex environment and must adapt when the environment changes. Optimization, which implies choosing a particular strategy and applying that strategy to its fullest extent, may not—and should not to be compatible with biological models—be a viable survival technique. Instead, agents should evolve a variety of strategies and the ability to apply them in appropriate situations.

The second issue concerns the search space available to agents: it should have as few constraints as possible. Our approach is not to provide agents with strategies but with the means to develop strategies. For example, one of the tasks that agents must perform is to maintain their energy balance. As designers of the experiment, we can see many ways of doing this. But if we build our preconceptions into the agents, the experiment suffers. Instead, we provide agents with the capability to perform simple tasks that in combination can be used to implement an energy balancing strategy. To find an appropriate strategy requires that the agents evolve.

We have been somewhat unsuccessful in our attempts to resolve the issue of anthropomorphism: anthropomorphic language is simply too effective and too tempting to avoid altogether. Although we speak amongst ourselves of agents ‘eating’, ‘moving’, ‘fighting’, and so on, we try to maintain a less anthropomorphic perspective on the software we are writing. The work of programming provides a useful ‘grounding’ experience. It is difficult to pretend that you are simulating ‘eating’ when you write code that moves a character from one array to another.

The aim of our project is to provide some insight into the Gould *versus* Wolfram debate. We seed the simulation (Section 4) with agents that have somewhat more complexity than is required to exist in the environment. According to Gould, the combination

of random effects produced by evolution and a lower bound to complexity should result in a slow increase in average complexity. According to Wolfram, evolution will select the simplest agents that can exist in the environment. Since our agents are automata and not merely optimizers, we expect our results to provide some insight as to which argument holds, at least in the context of our simulation.

4 The FormAL Simulation

The program simulates the behaviour of a population of *agents*. We distinguish between *laws* that are fixed by the simulation program and which are analogous to the laws of physics in our world, and *life*, which may evolve in any way that is consistent with those laws.

4.1 Laws

The design of the environment in which agents function is a critical part of the simulation. The environment must be sufficiently complex to enable evolution of interesting behaviour to occur but not so complex that the first generation of agents does not survive. The environment must obey laws that agents can discover during their evolution. If the environment behaves randomly, agents cannot learn from their experience—and, in fact, may develop superstitions! As a simple example, consider interactions between agents.

In many EP experiments, the simulation picks agents at random and allows them to interact. This implies that there is a ‘law of interaction’ and it states that interactions are random. Agents have no control over their interactions and cannot evolve behaviour concerning interactions, such as the frequency of interactions or the choice of partner. In our program, interactions are in fact determined by the simulation, but only agents that are close together are allowed to interact. This has several consequences.

- For “close together” to be meaningful, agents must exist in a metric space.
- Agents must have the ability to move in the space.
- Agents must have the ability to sense other agents.

The simulation must provide all of these features if agents are to evolve any kind of meaningful control over their interactions.

We have chosen to use a three dimensional space with a Euclidean metric. We decided to use three dimensions rather than the more usual two because 3D offers richer possibilities than 2D for interaction. The 3D space is continuous in the sense that coordinates are real numbers, but it is divided into a finite grid of

cubical *cells* to reduce the overhead of certain calculations. For example, agents can interact only if they are in the same cell; this avoids having to calculate the $\mathcal{O}(N^2)$ distances between N agents. The space is closed in each direction to avoid boundary effects: coordinate calculations are performed modulo M , where M is a preassigned maximum value for a dimension, giving the effect of a 3D torus embedded in 4D space. Other metric spaces could be used and might be more efficient (e.g., agent addresses with bit difference as distance) but 3D Euclidean space has the advantage of being easy for people to visualize.

The space is permeated with ‘atoms’, represented by the 26 lower case letters of the Roman alphabet. Atoms may have high or low energy. Some atoms are naturally plentiful and are appropriate sources of energy. Other atoms are scarce, and may be better suited to signalling and communication. We do not tell the agents what to do. We leave them to find out what works and what doesn’t. The ‘laws’ ensure that the total number of atoms is conserved. The simulation feeds energy into the environment by converting low-energy atoms to high-energy atoms.

4.2 Life

An agent consists of several *parts*, each of which is a cube. Parts can be joined together by sharing a face. Consequently, a part can have up to six neighbours. The accessibility of a part depends on how many neighbouring parts it has: a part with no neighbours is fully accessible; a part with six neighbours is inaccessible. Accessibility can be thought of as distinguishing external and internal organs of a body: a sensor requires accessibility but a digester does not.

An agent is created as a single part that is separated from another agent. A newborn agent grows by dividing into parts. When it has acquired several parts, it can split off one of the parts, yielding a new agent. This is analogous to the birth of a child that started life as a part of its parent. An agent may also split into two agents each consisting of several parts, and two agents may combine into a single agent. Thus there are various techniques for increasing complexity and for reproducing, and we rely on evolution to discover which are the most effective.

Agents extract energy from the environment by obtaining high-energy atoms, converting them to low-energy atoms, and returning low-energy atoms to the environment.

It is clear that the behaviour of an agent can be complex and that the *range* of possible behaviours is extensive. Following the conventional biological model, the behaviour of the agent is encoded in its genome. But how can a genome express such a wide range of behaviour? One solution is to make the genome a program in a simple programming language.

c	Convert atoms to obtain energy
d	Reproduce by cloning
e	Regulate energy
f	Divide this agent into two
g	Get atoms from environment
h	Convert atoms for communication
i	Sense mood of interacting agent
j	Interact with another agent
m	Regulate mass
p	Put atoms into environment
q	Reproduce with mutation
r	Reproduce using crossover
s	Sense atoms in the environment
u	Sense agents in the environment

Table 1. FormAL genes

Agents cannot modify the effect of an instruction in the language; in fact, they cannot even modify their own programs. All they can do is employ the machinery of evolution to discover programs that solve the problems necessary for survival. Systems of this kind are called *auto-adaptive*.

FormAL lies somewhere between ‘genome as descriptor’ and ‘genome as program’. The genome of an agent is a string of atoms (that is, letters). The genome is divided into *genes*, where each gene is a group of three letters, also called a *triplet*. In the triplet *pga*: *p* is the probability that the gene will be activated during a given step; *g* determines the effect of the gene; and *a* is an argument that may be used to modify the effect. Thus there are 26 distinct effects, each with 26 variations, giving an effective vocabulary of 676 actions. The genome is of arbitrary length and may contain several genes with related effects. The potential behaviour space is consequently quite large.

That a gene has an effect makes the genome similar to a program. However, genes can communicate their effects only indirectly, and their execution is not fully deterministic. In this respect, genomes are not like conventional programs.

Table 1 shows the genes that are used in the current version of the program. These genes express very complex behaviours (macro-behaviours). As the program develops, we will break these genes down into much simpler ones that express micro-behaviours. Gene *g* allows an agent to obtain atoms from the environment. Clearly, it makes sense to look for high-energy atoms that can be converted to low-energy atoms (gene *c*) but this behaviour must be evolved, not built-in. Various forms of reproduction are allowed, leaving agents to decide which is the most effective.

There is no explicit fitness function but agents that do not perform any significant actions for a sus-

tained period of time are condemned to death. Despite the weakness of this selection force, the system consistently converges. The simulation starts with a population containing several hundred genomes; after it has run for a while (typically 10,000 – 50,000 time steps), the population consists of several thousand clones — agents with the same genome.

5 Related Work

The idea of using a programming language to express behaviour is not new [1, 12, 14]. The languages used by **tierra** and **avida** can be loosely categorized as ‘von Neumann’ languages because they are based on the idea of a processor that executes instructions and stores results in a memory. There is also a large body of work that is loosely based on Church’s λ -calculus [4], including most of Koza’s detailed studies [9, 10], Fontana’s ‘Algorithmic Chemistry’ [6], and Rasmussen *et al*’s ‘Dynamical Hierarchies’ [13].

The languages used in **tierra** and **avida**, however, are so similar to conventional programming languages that their usefulness in studies of evolution is limited. Their drawbacks include instructions that: require unbounded data, such as stack operations; do too much, such as ‘replicate the agent’; are uninteresting and unrealistic in a biological sense, such as arithmetic instructions; are ‘brittle’, so that a small change has large effects. Accordingly, we have tried to design a language in which it is possible to describe the kind of actions that an agent might perform while avoiding the ‘brittleness’ of typical programming languages. This has proved to be surprisingly difficult.

Although we are not primarily interested in optimization, our work bears some relation to recent studies in “multi-objective optimization” such as [5].

6 Applications

A significant advantage of the view that evolutionary programs solve optimization problems is that there are many optimization problems to be solved. In an optimization problem, the population of individuals is an artifact of the solving process only; the solution itself consists of a unique individual whose genome encodes optimal parameter values.

In FormAL, the population is a part of both the process and the solution itself. Applications of a population of evolving agents are less obvious than those of a unique, optimal agent. There are, however, many examples of programs that are run as multiple copies. Apart from the obvious examples—most PC owners run the same common core suite of programs—there are also programs, such as packet switchers and routers, that exist in many places, run autonomously, and yet can communicate with one another. In most

cases, all instances of the program are identical. There is no need for this to be so, provided that all instances perform the basic tasks that they are required to perform. Such a collection of programs could be considered as an evolving population. All that is needed are fitness criteria, a genetic description, and the ability to replace inferior versions by better versions of the program. There are already many programs in this category, and as networks become denser and more ubiquitous, the number is sure to grow.

One family of programs most likely to benefit from evolutionary techniques is viruses. There is not much that we can do about that. Viruses and virus prevention techniques will participate in an inevitable arms race, thereby demonstrating the phenomenon, well-known to EP researchers, that the best way to stimulate evolution is to provide an enemy.

7 Conclusion

We identify four issues that have made EP research less productive than it might be: optimization provides a very restricted view of evolution; the definition of the problem space prejudices our assessment of the results; imprecise terminology leads to confusion; and it is crucial to realize that in evolution complexity and simplicity are not mutually exclusive. The **FormAL** simulation provides an environment in which we can investigate these issues and their implications.

Acknowledgments The research described in this paper was funded by the National Science and Engineering Research Council of Canada. Dr. Nawwaf Kharma and Taras Kowaliw have attended many of our meetings and we are grateful to them for their valuable suggestions and advice.

References

- [1] C. Adami and C.T. Brown. Evolutionary learning in the 2D artificial life system **avida**. In R. Brooks and P. Maes, editors, *Artificial Life IV*, pages 377–381. MIT Press, 1994.
- [2] Aristotle. *Categories*. Publisher unknown, –350.
- [3] George Boole. The calculus of logic. *Cambridge and Dublin Mathematical Journal*, III:183–98, 1848.
- [4] A. Church. *The Calculi of Lambda Conversion*. Princeton University Press, 1941.
- [5] David W. Corne, Kalyanmoy Deb, Peter J. Fleming, and Joshua D. Knowles. The good of the many outweighs the good of the one: Evolutionary multi-objective optimization. *IEEE Connections*, pages 9–13, February 2003.
- [6] Walter Fontana. Algorithmic chemistry. In [11], pages 159–209, 1992.
- [7] Stephen Jay Gould. *The Structure of Evolutionary Theory*. Harvard University Press, 2002.
- [8] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975. Second Edition, MIT Press, 1992.
- [9] John R. Koza. *Genetic Programming: on the Programming of Computers by Means of Natural Selection*. Complex adaptive systems. MIT Press, 1992.
- [10] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. Complex adaptive systems. MIT Press, 1994.
- [11] Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors. *Artificial Life II*, volume X of *Santa Fe Institute: Studies in the Sciences of Complexity*. Addison Wesley, 1992.
- [12] C. Ofria, C. Adami, and T.C. Collier. Design of evolvable computer languages. *IEEE Trans. Evolutionary Computation*, 6(4):420–4, August 2002.
- [13] Steen Rasmussen, Nils A. Baas, Bernd Mayer, Martin Nilsson, and Michael W. Olesen. *Ansatz for dynamical hierarchies*. *Artificial Life*, 7:329–353, 2001.
- [14] Thomas S. Ray. An approach to the synthesis of life. In [11], pages 371–408. Addison Wesley, 1992.
- [15] Guy Sella and David H. Ardell. The coevolution of genes and the genetic code. Technical Report 01-03-015, Santa Fe Institute, February 2001. Available from <http://www.santafe.edu>.
- [16] C.H. Waddington. Discussion. In P.S. Moorehead and M.M. Kaplan, editors, *Mathematical Challenges to the Neodarwinian Interpretation of Evolution*. Wistar Institute, Philadelphia, PA, 1967. Cited in [7], page 584.
- [17] Günter Weberndorfer, Ivo L. Hofacker, and Peter F. Stadler. On the evolution of primitive genetic codes. Technical Report 02-08-034, Santa Fe Institute, 2002. Available from <http://www.santafe.edu>.
- [18] Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002.