



## Simulating Evolution with *Mathematica*

C. Jacob, jacob@cpsc.ucalgary.ca,

*Department of Computer Science, The University of Calgary,  
2500 University Drive N.W., Calgary, Alberta, Canada T2N 1N4*

### Abstract

Evolutionary mechanisms as observed in nature are successfully used in evolutionary algorithms (EA) in order to solve complex optimization tasks or to mimick natural evolution processes. We present a collection of evolutionary algorithms which we have implemented in *Mathematica* together with some visualization examples and applications. The three major EA-classes are discussed: Evolution Strategies (ES), Genetic Algorithms (GA), and Genetic Programming (GP). Interactive evolution is demonstrated by the breeding of biomorphs, recursively branched line drawings. Multi-modal ES- and GA-experiments are demonstrated for a parameter optimization task. The evolution of robot control programs shows a simple GP-application. The article concludes with a more sophisticated GP-example: the breeding of developmental programs for artificial plantlike structures encoded on the basis of Lindenmayer systems.

### 1 Evolutionary algorithms – a short introduction

Evolution in nature provides fascinating mechanisms for adapting living species to their environments. It is somewhat remarkable that such complex adaptations, taking place over huge time spans (from hundreds to millions of years) are essentially driven by a strikingly simple principle: iterated selection and mutations, as originally formulated by Charles Darwin (Darwin [2]). All organisms that are able to reproduce, thus transmitting their specific genes to their descendants, have won nature's implicit »struggle-of-survival« test. The genes are passed on to the next generation, however, with some minor mutations, either due to imperfect copying of the genetic information or due to recombination effects observable among sexually reproducing individuals.

With this natural scenario of adaptation in mind, why shouldn't evolutiona-

## 264 Innovation In Mathematics

ry concepts make up a promising technique for search and optimization algorithms? Trying to integrate evolution into algorithmic schemes, is just what many researchers, propagating the field known today as »evolutionary computation« (EC), have been doing for quite a long time (Holland [6], Rechenberg [18]). Consequently, many different versions and applications of evolutionary algorithms (EAs) have been developed, most of them belonging to one of three major EA-classes, we will discuss in the sequel of this article: Genetic Algorithms (GA), Evolution Strategies (ES), and Genetic Programming (GP).

### An evolutionary algorithm sketch

All the representatives of the EA paradigm we would like to discuss here are instances of a general framework of simulated evolution. What are the basic ingredients of an evolutionary algorithm? Most evolution-based algorithms perform according to the following scheme:

1. Generate an initial *population* of *structures* (»chromosomes«) encoding some genotypic information.
2. Interpret each structure and assign it a *fitness* value.
3. If a termination criterion is satisfied: STOP.
4. Declare which of the structures are to *survive* for the next generation, depending on the individuals' fitnesses.
5. Create variants of the survivors by *mutation* or *recombination*, which constitute the following population.
6. Continue with step 2.

What makes the difference among the diverse approaches of evolutionary algorithms, is first of all the way in which they represent their genotype structures. These genotypic encodings range from real-valued vectors (ES) or strings over a discrete (usually binary) alphabet (GA) to symbolic expressions (GP) that even allow hierarchically structured computer programs to be evolved. Secondly, evolutionary algorithms can be classified with respect to the operators they use to mutate their chromosomes. These operators, most of them gleaned from natural gene mutation mechanisms in real cells, range from simple point mutations to recombinations, mimicking the interchange of genes among several individuals. Another characteristic EA-property is how individuals are selected from generation to generation, answering the question of »who shall live and who shall die«.

Although these evolutionary algorithms are different in some way or another, they still have so much in common (as, e.g., the EA scheme mentioned above) that we are able to implement them in a quite uniform way, as we have been doing in *Mathematica*. The following paragraphs will show some application examples.

## 2 From interactive breeding to evolutionary programming

The following chapters will present some *Mathematica* implementation examples of evolutionary algorithms: an interactive breeding algorithm demonstrated by the evolution of »biomorphs«, the use of Evolution Strategies to solve an optimization task with multiple sub-populations, an example for visualizing the dynamics of binary genotypic structures used with Genetic Algorithms, how to animate the solutions of a Genetic Programming robot control task, and, finally, the evolution of plant structures by breeding developmental programs encoded as Lindenmayer systems.

### 2.1 Breeding biomorphs – a simple selection-mutation scheme

Two-dimensional sketch figures, known as biomorphs, have been introduced by Richard Dawkins in order to demonstrate the creative potential of simple selection-mutation schemes (Dawkins [3, 4]). Tree-like, recursively constructed figures are described by a vector of 21 parameters, where, e.g., the first one gives the recursion depth, and all following numbers describing color, thickness, and length of each arch, as well as its branching angle and the number of branches, just to mention a few (figure 1).

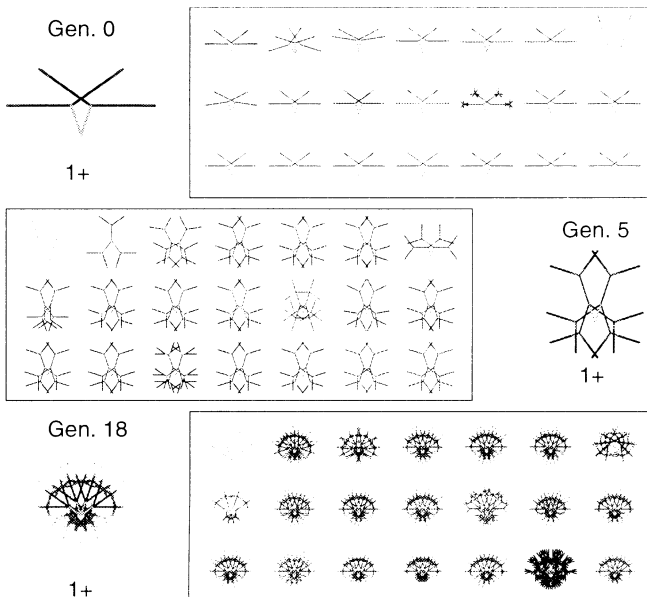


Figure 1: Interactive breeding of biomorphs

## 266 Innovation In Mathematics

Starting with an initial population of randomly generated biomorph vectors, we get structures similar to those shown in figure 1 (Gen. 0). Now if we select one of those biomorphs which we like most for some reason or another as, for instance, the one shown left to the population, and subsequently generate slightly mutated versions of this dedicated breeder biomorph's genotype, we arrive at yet another population of biomorphs, the next generation (not shown in figure 1). Continuing this selection-mutation interaction, we are finally able to create quite complex looking structures, as compared to the samples we started with (figure 1, Gen. 18).

### 2.2 Evolution Strategies

Evolution Strategies (ES) have been invented during the 60s and 70s as a technique of evolutionary experimentation for solving complex optimization problems, mainly within engineering domains (Rechenberg [18, 19], 1994, Bäck [1]). The preferred EA data structures are vectors of real numbers. Specially tailored mutation operators are used to produce slight variations on the vector elements by adding, with a certain probability, normally distributed numbers (with mean zero) to each of the components. Recombination operators have been designed for interchanging or mixing of »genes« between two or more vectors. These recombinations range from simply swapping respective components among two vectors to componentwise computation of means.

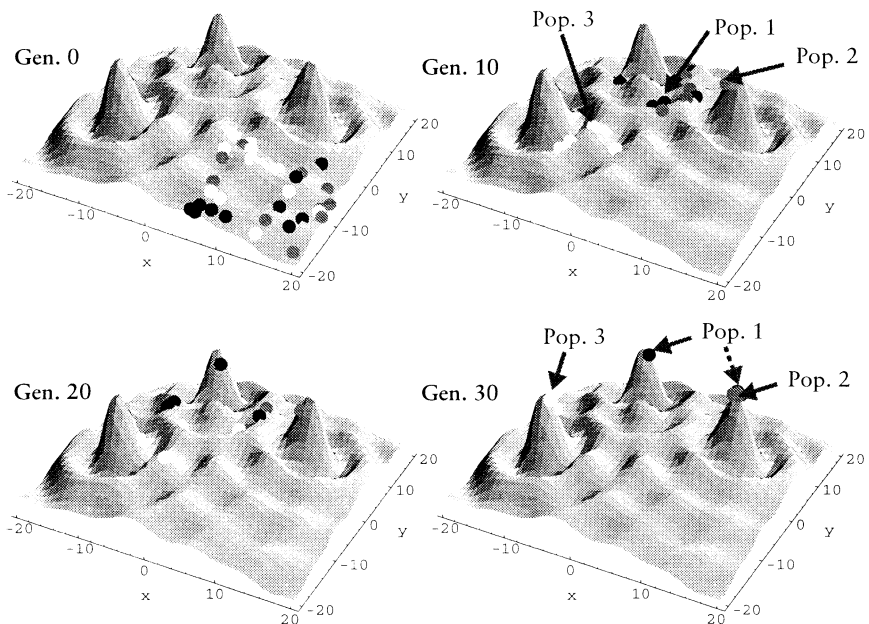


Figure 2: Evolution Strategies: Three populations finding their way to the top

Evolution Strategies have developed sophisticated methods of selection, especially when several sub-populations are involved in the evolution scheme. Figure 2 gives a typical example: Three populations, marked as white, gray and black spots, have to independently search for locations of maximum height within their »environment«. The x- and y-coordinates are encoded on the ES-vectors, with each population consisting of 10 members. Initially the individuals are randomly spread within the lower right segment (figure 2 a). After 10 generations of iterative selection, mutation and recombination all three populations have evolved to higher regions, some individuals being even on their way to the top (figure 2 b). Finally, after 30 generations all populations have obviously converged to one of the major viewpoints of the »mountain region« (figure 2 c), showing that Evolution Strategies are capable of solving multimodal optimization problems.

## 2.3 Genetic Algorithms

Genetic Algorithms (GA) as originally introduced by John Holland [6, 7] have been popularized as universal optimization algorithms based on evolutionary methods (see also Goldberg [5], Bäck [1]). The most remarkable difference between Evolution Strategies and Genetic Algorithms is that GAs use a stringbased, usually binary parameter encoding, resembling the discrete nucleotid coding on cellular chromosomes. Therefore, GA-genotypes consist of bit-strings, on which point mutations are defined by flipping bits with a certain probability. The recombinational operator of »crossover« models the breaking of two chromosomes and subsequent crosswise restitution. Each individual is assigned a fitness value depending on the optimization task to be solved. Selection within the population is performed in a fitness-proportionate way: The more fit the more likely an individual is to be chosen for reproduction into the following generation.

Figure 3 gives an example of an evolution sequence based on Genetic Algorithms. The optimization task is similar to the one in the previous chapter; the location of the global maximum has to be found. Coordinates are now encoded as binary strings of length 20, with the first half encoding for the x-coordinate, and the remaining bits representing the y-coordinates. Besides the phenotypic view of the evolution of 40 individuals per generation, only the best quarter of which is actually depicted, the black-and-white patterns show the development of the genotypes as matrices of binary vectors. Each »0« is represented by a black square, each »1« shown as a white square. In order to be able to easily recognize any emerging patterns the matrix plots have been smoothed. It can be clearly seen that the initial genotypes have been randomly generated, no clear pattern is recognizable for the genotypes (figure 3, Gen. 1). However, after some generations, and definitely at generation 10, stripes have obviously emerged showing that the genotypic strings have converged, and so have their corresponding phenotypes: They tightly gather around the global maximum.

## 268 Innovation In Mathematics

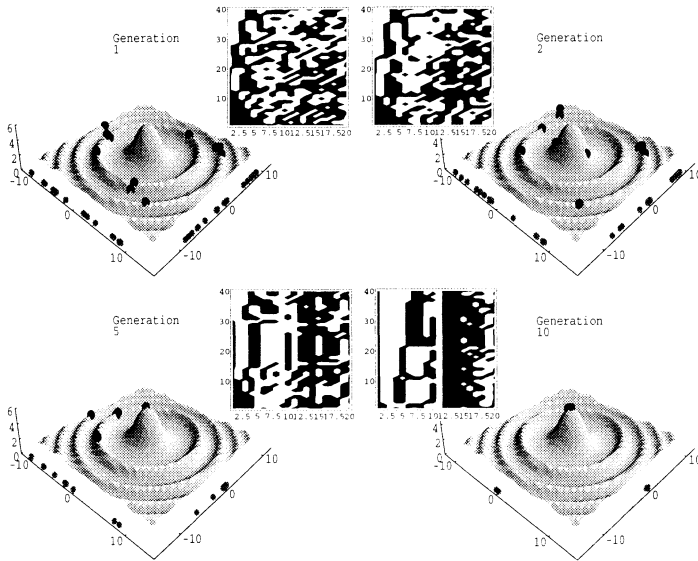
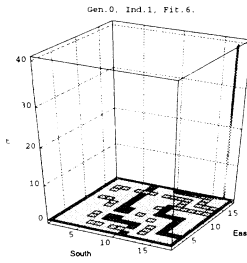


Figure 3: Genetic Algorithms: Visualizing the development of phenotypes and their corresponding binary-string genotypes

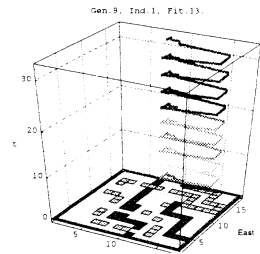
## 2.4 Genetic Programming

Genetic Algorithms serve well for linear, string-like encodings. However, if computer programs have to be evolved instead of »flat« parameter vectors or strings, a different representation scheme has to be found. As all *Mathematica* users know, symbolic expressions are a very good means to formulate computer programs in a uniform way. The uniformity of encodings is especially important for the implementation of genetic operators performing variations on the program structures. This is why symbolic expressions serve as a popular encoding within Genetic Programming (GP) systems, an extension of Genetic Algorithms with binary strings replaced by tree-like expressions (Koza [14, 15]).

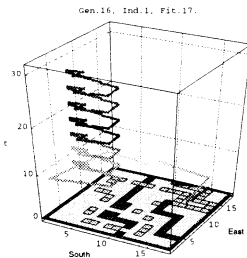
The following brief example should demonstrate what GP is about. Suppose an »artificial ant« is placed within a two-dimensional grid area as depicted in figure 4. The dark black squares represent walls, the gray squares spread like a trace are »food pieces« the ant will have to pick up. Initially, the ant sits at position (12,15) heading east. The ant is able to sense the field it is facing to, that is, the ant has got sensors that can especially tell food squares from wall pieces. At one action step the ant can either move forward to the next square (if it is not in front of a wall) or turn left or right for 90 degrees.

**Gen. 0**

```
AntTracker[
  advance
]
```

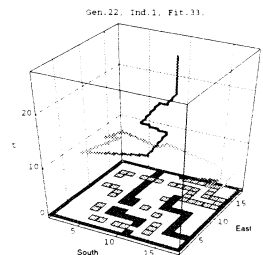
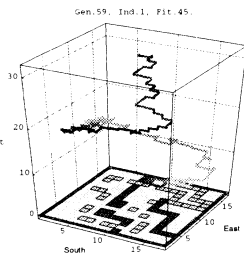
**Gen. 9**

```
AntTracker[seq[advance,
  ifSensor[dust][seq[stop], advance],
  ifSensor[dust][
    seq[turnLeft, again], nop], advance,
  ifSensor[wall][
    seq[turnLeft, again], advance]]]
```

**Gen. 16**

```
AntTracker[seq[advance,
  ifSensor[wall][seq[stop], turnRight],
  ifSensor[dust][seq[turnLeft, again], nop],
  ifSensor[wall][
    seq[turnLeft, again], advance]]]
```

```
AntTracker[seq[
  ifSensor[wall][seq[turnLeft, again],
    seq[turnRight]],
  seq[ifSensor[dust][
    seq[turnLeft, again], seq[nop]],
  seq[
    ifSensor[wall][
      seq[seq[turnLeft, again], seq[nop],
        ifSensor[phero][seq[stop]]],
      seq[seq[advance]]]], seq[nop]]]]]
```

**Gen. 22****Gen. 59**

```
AntTracker[seq[nop,
  ifSensor[wall][
    seq[turnLeft, turnLeft, again],
    seq[nop,
      ifSensor[wall][
        seq[turnLeft, again], turnRight],
      ifSensor[wall][
        seq[turnLeft, again], advance]]],
  ifSensor[dust][seq[turnLeft, again]],
  ifSensor[wall][
    seq[ifSensor[dust][advance]],
    seq[ifSensor[dust][seq[turnLeft, again]],
      ifSensor[wall][seq[turnLeft, again],
        advance]]]]]
```

Figure 4: Genetic Programming: Evolving food-foraging programs

## 270 Innovation In Mathematics

The optimization task is to evolve control programs that enable the ant to reach all food squares within a given time limit. Figure 4 additionally gives examples of control programs that have been evolved by Genetic Programming methods. Shown are the best-of-generation programs together with their according action diagrams visualizing the ant's movements in time ( $t$ ). We again started with randomly generated program structures, 50 per population. For the first generation (Gen. 0) a very simple program turns out to be best: The ant moves forward until it is stopped in front of the wall towards which it is heading. Some generations further periodic, circular movements of the ant evolve (Gen. 9 and 16). In generation 22 the ant follows the food trail nearly perfectly, except for the beginning where it misses several food squares. Finally, after 59 generations a program has been evolved that implements an ant not only following the given trail but also being able to actively search for food in off-trail regions.

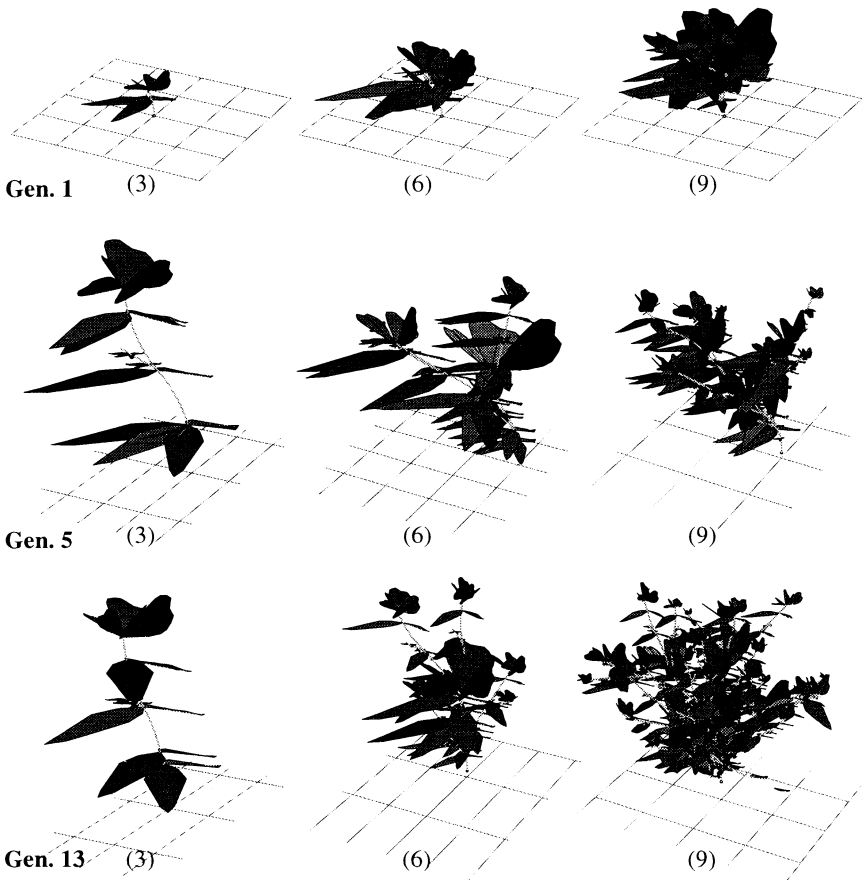


Figure 5: Simulated evolution of plant-like structures encoded as L-systems

## 2.5 Genetic L-system programming

Parallel rewrite systems in the form of stringbased Lindenmayer systems (Lindenmayer [16]), or L-systems for short, have been successfully used for modeling and visualizing growth processes of biological organisms, especially artificial plants (Prusinkiewicz & Lindenmayer [17]). We have been implementing a package for context-sensitive  $(m,n)$ -L-systems which takes full advantage of *Mathematica*'s expression manipulation and graphics capabilities (Jacob [9, 10]).

Formulating growth programs by Lindenmayer systems and encoding these rewrite systems as symbolic expressions gives us the chance to simulate evolution of morphogenetic processes on the basis of Genetic Programming. Figure 5 shows very simple examples of GP-evolved »plants«. Each of these plantlike structures are represented by L-systems, i.e. their pictures shown are not static but can be animated, thus visualizing the plants' morphogenesis. In this example fitness is calculated as plant complexity with respect to the number of branches, leaves and blooms, as well as the plant's extension in width and height. Figure 5 depicts growth stages of some best-performing plant structures evolved over several generations. The graphics are scaled such that most of the details are visible. The numbers in brackets denote L-system iterations. From initially simple looking plants we finally arrive at rather complex structures after only a few generations.

More detailed descriptions of the Genetic L-System Programming paradigm can be found in (Jacob [8, 11, 12]). Currently this system is being extended for the simulation of co-evolutionary processes.

## Conclusions

We have demonstrated by several examples how different classes of evolutionary algorithms – Evolution Strategies, Genetic Algorithms, and Genetic Programming – can be implemented and visualized with *Mathematica*. All examples shown here as well as all EA-implementations are available as *Mathematica* notebooks. Most of the graphics shown are part of animation sequences showing the populations evolving.

Thanks to *Mathematica*'s excellent graphics capabilities, those and similar evolution movies have turned out to be of great help to explain and capture the basic mechanisms of evolution processes and evolutionary algorithms in particular. The software has been successfully used in several student courses.

A selection of EA-related notebooks is also available via the World Wide Web: <http://www2.informatik.uni-erlangen/IMMD-II/Persons/jacob> or alternatively <http://www.cpsc.ucalgary.ca/~jacob>.

A more indepth treatment of evolutionary algorithms with *Mathematica* can be found in the forth-coming book *Principia Evolvica* (Jacob [13]).



## References

1. Bäck, T., *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, Oxford, 1996.
2. Darwin, C., *On the origin of Species*, Murray, London, 1859.
3. Dawkins, R., *The Blind Watchmaker*, Longman, Oxford, 1986.
4. Dawkins, R., *Climbing Mount Improbable*, Norton & Company, New York, 1996.
5. Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Redwood City, 1989.
6. Holland, J. H., *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor, Mich., 1975.
7. Holland, J. H., *Adaptation in Natural and Artificial Systems*, The MIT Press, Cambridge, 1992.
8. Jacob, C., *Genetic L-System Programming*, in: Parallel Problem Solving from Nature - PPSN III, Lecture Notes in Computer Science 866, Springer-Verlag, Berlin, 1994.
9. Jacob, C., *Genetic L-system programming: breeding and evolving artificial flowers with Mathematica*, in: V. Keränen, P. Mitic (eds.), *Mathematica with Vision (IMS'95)*, Computational Mechanics Publications, Southampton, 1995.
10. Jacob, C., *Modeling Growth with L-Systems & Mathematica*, in: *Mathematica in Education and Research*, Vol. 4, No. 3, TELOS Springer, 1995.
11. Jacob, C., *Evolving Evolution Programs: Genetic Programming and L-Systems*, in: J. Koza, D. Goldberg, D. Fogel, R. Riolo (eds.), *Genetic Programming 1996*, The MIT Press, Cambridge, MA, 1996.
12. Jacob, C., *Evolution programs evolved*, in: Parallel Problem Solving from Nature - PPSN IV, Lecture Notes in Computer Science 1141, Springer-Verlag, Berlin, 1996.
13. Jacob, C., *Principia Evolvica – Evolutionäre Algorithmen mit Mathematica*, dpunkt.Verlag, Heidelberg, 1997.
14. Koza, J., *Genetic Programming*, MIT-Press, 1992.
15. Koza, J., *Genetic Programming II*, MIT-Press, 1994.
16. Lindenmayer, A., *Mathematical models for cellular interaction in development, Parts I and II*, *Journal of Theoretical Biology*, 18:280-315, 1968.
17. Prusinkiewicz, P., and Lindenmayer, A., *The Algorithmic Beauty of Plants*, Springer-Verlag, New York, 1990.
18. Rechenberg, I., *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Frommann-Holzboog, Stuttgart, 1973.
19. Rechenberg, I., *Evolutionsstrategie'94*, Werkstatt Bionik und Evolutionstechnik, Band 1, Frommann-Holzboog, Stuttgart, 1994.