Book review

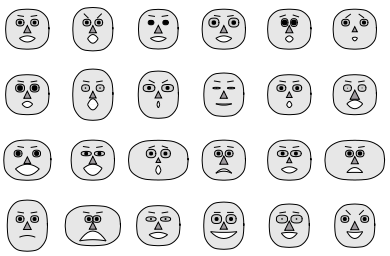
Marc Ebner
Universität Würzburg, Lehrstuhl für Informatik II
Am Hubland, 97074 Würzburg, Germany
ebner@informatik.uni-wuerzburg.de
http://www2.informatik.uni-wuerzburg.de/staff/ebner/welcome.html

November 6, 2002

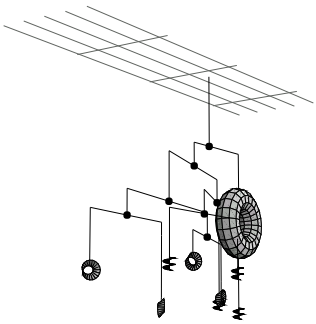
"Illustrating Evolutionary Computation with Mathematica" by Christian Jacob, Morgan Kaufmann Publishers, San Francisco, CA, 578 pages, 265 figures, 8 color plates and 96 programs, ISBN 1-55860-637-8, hard cover, list price $69.95.

"Illustrating Evolutionary Computation with Mathematica" is a nice introduction into the field of evolutionary algorithms. Its main focus is on illustrating evolutionary algorithms. Jacob unleashes the power of Mathematica to illustrate evolution in action. The book starts with a short introduction which describes the main ingredients of evolutionary algorithms. How evolution works is shown on the problem of evolving the sentence "Evolution of Structure, Step by Step". Mutation and selection are used to evolve the sentence from a random string. Mutation randomly changes some of the characters and selection only retains those strings which match the target string best. It is a simple problem yet it explains the main ingredients of evolution. Mathematica is used to illustrate how the population adapts to the target string over time. Visualization of an evolutionary algorithm is usually done by calculating minimum, average, and maximum fitness of the population. The values are plotted on a graph which shows fitness over time. Other statistics which might be reported are the standard deviation, diversity measures of the population or convergence measures. In his book, Jacob shows how Mathematica can be used to visualize what is happening during an evolutionary run. Using Mathematica this can be done easily no matter if one wants to visualize a single genotype, a population of genotypes, plot one- or two-dimensional functions or even use three-dimensional graphics.

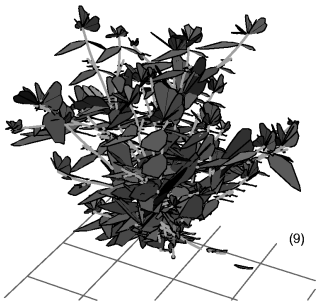
For the introductory problem of evolving a sentence, the population of individuals is shown with gray blocks marking those letters which are still incorrect or using intensity values to visualize Hamming distance. Apart from the evolution of a sentence, evolution of the color of butterflies and Dawkins' [2] biomorphs are also used as introductory problems. For the butterfly problem



Part of Figure 3.21
page 122



Part of Figure 7.26
page 395



Part of Figure 11.21
page 518

each individual represents a butterfly of a certain color. A single floating point value is used to specify the color of the butterfly. The population of butterflies needs to adapt to the environment because those who have a color different from the background are more likely to be eaten by a predator. Here Mathematica is used to draw a stylized icon of the butterfly using the floating point value as the color of the butterfly. Dawkins' biomorphs are visualized using line drawings.

After the introduction, Jacob gives a formal model of evolution. Following that, genetic algorithms [7, 6] are discussed in detail. Representation issues including binary and real-valued chromosomes, diploid and m-ploid chromosomes, dominance and the biological alphabet are discussed first. The genetic operators mutation, recombination, inversion, deletion and duplication are described next. Deletions are handled by selecting a random subsequence and removing it. Duplications are handled by selecting a random subsequence and inserting it again right after the selected sequence. Obviously the genotype shrinks, respectively grows, if these operators are applied. Decoding is done by partitioning the genotype into as many segments as there are variables. The following selection methods are covered: fitness-proportionate section, rank-based selection and elitist selection. Using Mathematica Jacob shows the effects of the use of a particular set of operators has on the population. The genetic material of the population as well as the movement on the fitness landscape is visualized. The schema theorem and its problems are discussed at the end of the chapter. A highlight of this chapter is the introduction of Chernov figures to visualize genotypes (Fig. 3.21). A Chernov figure is a stylized face with eyes, eye-brows, nose and mouth. The allele values are used to vary the shape of the head, the shape of the eyes, orientation of the eye-brows, the shape of the mouth and so on. The human ability to recognize faces helps to compare different genotypes visualized using Chernov figures.

An introduction to evolution strategies is given next. Representation of individuals, mutation and recombination, different selection and reproduction schemes as well as step size adaptation are described in detail, correlated mutations, however, are not covered. For correlated mutations the reader is referred to additional literature. Mathematica is again used to visualize the evolving population on its fitness landscape.

The second part of Jacob's book focuses on the evolution of computer programs. Again, a short introduction into the field is given, followed by a detailed introduction into evolutionary programming [4, 3] and genetic programming [12, 13, 1, 14]. Classifier systems are described briefly. Fogel et al. [5] originally coined the term evolutionary programming. It was used to describe the evolution of finite state machines. Today, the term evolutionary programming is used for a wide variety of evolutionary methods with many different representations [4, 3]. The chapter on evolutionary programming focuses only on the evolution of finite state machines. Jacob gives a short introduction into finite state machines and discusses the genetic operators used to modify them. Again, Mathematica is used to visualize the finite state machines and their output.

The chapters on genetic programming are a little difficult to read. Jacob makes a distinction between symbolic expressions and term structures. A term structure is anything which can be constructed using a set of elementary functions and terminal symbols. In a symbolic expression the function symbols may be terms themselves. Let's have a look at an example given in his book. The Mathematica expression `Derivative[2][Sin][pi]` computes the second derivative of the sine function evaluated at position π (p. 349). In LISP the same expression could be written as `(((Deriv 2) Sin)` π `)` (p. 293). That is, the parameter `2` specifies that we want a function computing the second derivative. This function is then applied to the term `Sin`. The second derivative of `Sin` is `-Sin`. Finally, this function is evaluated at position π which gives zero. In standard tree based GP the symbolic expression `Derivative[2][Sin][y+z]` might be represented as `(expr (expr (expr Derivative 2) Sin) (expr (+ y z)))` (p. 366) where expr is a function which evaluates to a function according to the arguments specified. The set of elementary functions is $F = \{$`expr`$\}$ and the set of terminal symbols includes `Deriv`, `Sin`, `2` and π. With this representation use of typed genetic programming is necessary. Otherwise it might happen that a function which requires an argument replaces a terminal symbol. Jacob's system uses selective GP recombination and mutation operators to ensure that only valid structures are created.

Another possible representation would be to use F= { `Eval`, `Deriv`, `+`, `-`, `*`, `/`, ... } and T= {

1, 2, y, z, ... }. This would be a much more intuitive representation. In this case `Eval` would be a function with arity two. The first argument would be a floating point value, the second argument would be a function. Thus, the function `Eval` would evaluate the given function at a given location. The function `Deriv` could be a binary function which computes the first derivative. The first argument specifies the variable for which the derivative will be computed, the second argument specifies the function to compute the derivative of. Alternatively `Deriv` could be a ternary function where the third argument can be used to calculate the n-th derivative. Thus, the symbolic expression `Derivative[2][Sin][y+z]` could be written as `(Eval (Derivative 2 x (Sin x)) (+ y z))`. Rather than making a distinction between term structures and symbolic expressions to me it seems more important to make a distinction between typed genetic programming and untyped genetic programming. Typed genetic programming is not treated in detail and the reader is referred to additional literature.

Genetic programming is illustrated on the problem of evolving balanced mobiles (Fig. 7.26). Mobiles are drawn in two or three dimensions using Mathematica's visualization capabilities. Here, symbolic expressions have the form `s[armLength1,armLength2][subMobile1,subMobile2]` where `armLength1` and `armLength2` specify the length of the two arms, `submobile1` and `submobile2` are again symbolic expressions. The recursion stops if a geometric shape is chosen as a subexpression. To actually evolve the mobiles, the expressions are converted into a term structure of the form `sik(subMobile1 subMobile2)` where `i` is the length of the first arm and `k` is the length of the second arm. Only four possible lengths are allowed.

A more standard notation would be to use $F = \{\text{Arm}, \text{Sphere}, \text{Cone}, ...\}$ and $T = \{R\}$. The elementary function Arm would be a four-argument function. The first argument specifies the length of the first arm, the second specifies the length of the second arm, and arguments three and four specify the substructures hanging on the first and second arm respectively. The elementary functions Sphere and Cone specify geometric elements. They take one argument each which specifies the weight of the geometric element. To evolve correct expressions typed genetic programming must be used.

The second problem used to illustrate how genetic programming works, is the evolution of a program which steers an artificial ant. The ant has to collect food pieces similar to the Santa Fe ant except that the ant also has to avoid walls. For this problem, Mathematica is used to visualize the path taken by the ant. Automatically defined functions, automatically defined iterations and loops are discussed briefly.

In part three of the book, Jacob describes the evolution of developmental programs. Of course, Jacob is best known for his work on the evolution of artificial plants using Lindenmayer systems (L-systems) [8, 9, 10, 11]. Part three consists of three chapters. The first chapter starts with a short introduction into cellular automata and Langton's self-reproducing loops. After that, L-systems [15] are described in detail. Context-free, parameterized L-Systems and context-sensitive L-Systems are covered. Examples include a a three-dimensional version of the Hilbert curve and a model of tree structures using bracketed L-Systems. The second chapter explains how L-Systems are encoded. Evolution of an L-System describing a quadratic Koch island is chosen as a sample problem. The final chapter of the book deals with the evolution of artificial flowers. The three-dimensional plants consisting of sprouts, stalks, leaves and blooms are visualized using Mathematica (Fig. 11.21). An entire plant ecosystem where different plant species compete with each other is also modeled. Here, a plant is modeled as a rosette on a stalk. The rosette models the size of the plant, its height is given by the length of the stalk.

A final note on the use Rechenberg's graphical notation to describe the structure of evolutionary algorithms. The graphical notation is used throughout the book as a unifying illustration of the different evolutionary algorithms. It appears first in the chapter on genetic algorithms. However, the notation is not introduced until the middle of the next chapter in the context of evolution strategies.

"Illustrating evolutionary computation with Mathematica" is largely easy and fun to read. I have had some difficulties with the chapters on genetic programming as described above. In conclusion, Christian Jacob has selected a set of interesting problems to explain how evolutionary algorithms work. It is thus suitable for university students as an introductory textbook into

the field. All main stream methods like genetic algorithms, evolution strategies, evolutionary programming (evolution of finite state machines) and genetic programming are covered. The end of each chapter contains biographical notes which give pointers to additional literature. The Mathematica notebooks for the book can be downloaded from the author's web page. The book shows step by step how Mathematica can be used for evolutionary computation. Mathematica notation is not explained. If one is not familiar with Mathematica one may be able to guess what the code is doing. However, readers who are not familiar with Mathematica are probably overwhelmed by some of the notation. A short introduction into Mathematica would be a great benefit for the book. This could be either integrated into the main text, i.e. whenever a new operator or symbol is used, its function is explained, or the introduction is given in an appendix of the book. Researchers in the field can learn from the book that it is important to visualize what your evolutionary algorithm is doing and that one can do a lot more than draw curves of the best, average and worst fitness of the population.

References

[1] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming - An Introduction: On The Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers, San Francisco, California, 1998.

[2] Richard Dawkins. *The Blind Watchmaker*. W. W. Norton & Company, New York, 1996.

[3] David B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, New York, 2000.

[4] Lawrence J. Fogel. *Intelligence Through Simulated Evolution: Forth Years of Evolutionary Programming*. John Wiley & Sons, New York, 1999.

[5] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. *Artificial Intelligence Through Simulated Evolution*. John Wiley & Sons, Inc., New York, 1966.

[6] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[7] John H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. The MIT Press, Cambridge, Massachusetts, 1992.

[8] Christian Jacob. Genetic l-system programming. In Yuval Davudor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature – PPSN III. The Third International Conference on Evolutionary Computation. Jerusalem, Israel, October 9-14*, pages 334–343, Berlin, 1994. Springer-Verlag.

[9] Christian Jacob. Evolution programs evolved. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV. The Fourth International Conference on Evolutionary Computation. Berlin, Germany, September 22-26*, pages 42–51, Berlin, 1996. Springer-Verlag.

[10] Christian Jacob. Evolving evolution programs: Genetic programming and l-systems. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Proceedings of the First Annual Conference on Genetic Programming*, pages 107–115, Cambridge, Massachusetts, 1996. The MIT Press.

[11] Christian Jacob. Evolution and coevolution of developmental programs. *Computer Physics Communications*, pages 46–50, 1999.

[12] John R. Koza. *Genetic Programming. On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, Massachusetts, 1992.

[13] John R. Koza. *Genetic Programming II. Automatic Discovery of Reusable Programs*. The MIT Press, Cambridge, Massachusetts, 1994.

[14] William B. Langdon and Riccardo Poli. *Foundations of Genetic Programming*. Springer-Verlag, Berlin, 2002.

[15] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer Verlag, New York, 1990.