– EVOLVICA –

Illustrating Evolutionary Computation

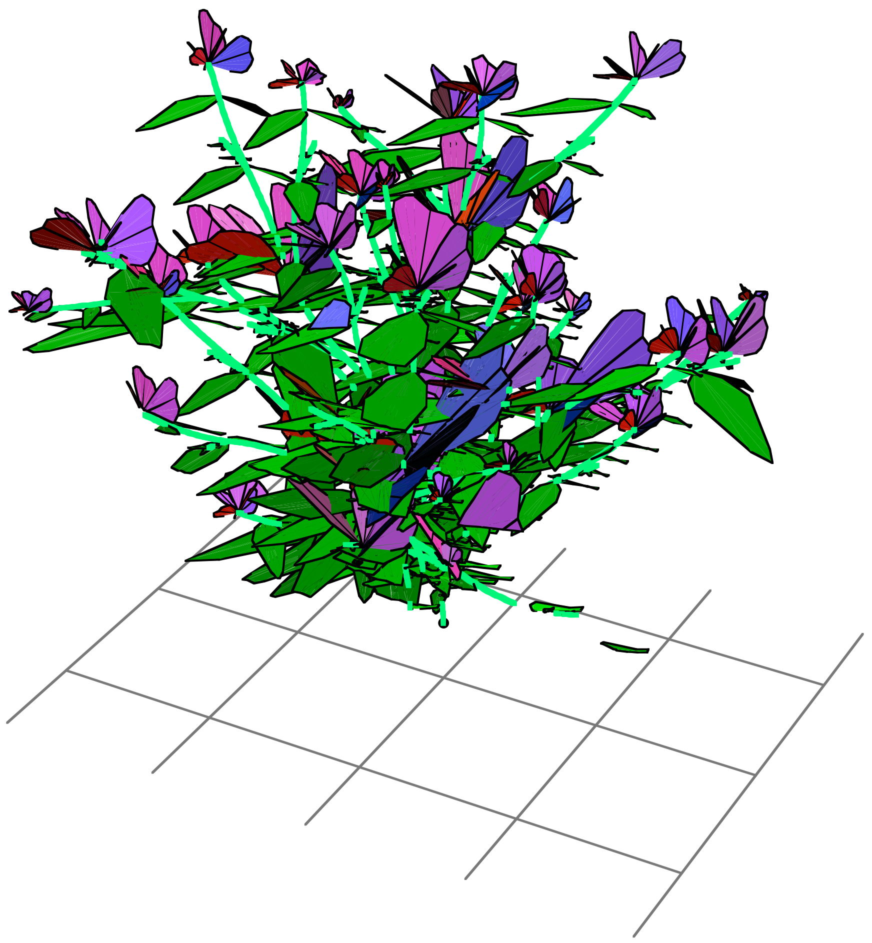
with Mathematica

© Christian Jacob

Department of Computer Science
The University of Calgary
CANADA

`jacob@cpsc.ucalgary.ca`
`http://www.cpsc.ucalgary.ca/~jacob/`

Morgan Kaufmann Publishers
San Francisco, CA
2001



See the flower grow ...

– EVOLVICA –

Keywords

Evolutionary algorithms, evolution strategies, genetic algorithms, genetic programming, cellular automata, Lindenmayer systems, evolution and co-evolution of developmental systems, models of competition and cooperation, complex adaptive systems, agent-based evolutionary computation

Table of Contents

1 Fascinating Evolution: Introductory Examples

This chapter demonstrates one of the fundamental principles of evolution: iterated selection and mutation by several simple, but yet – hopefully – convincing examples.

1.1 Cumulative Selection: "Me thinks it is like a weasel …"

How can iterated selection and random mutation help in finding a predefined sequence of characters?

1.2 Mimesis: Evolution of Butterfly Mimicry

Butterflies sitting on barks of trees adapt their wing colours to their environments. A simple simulation shows how this adaptation can be achieved through evolution.

1.3 Evolutionary Creativity: Biomorphs

On the basis of "biomorphs", simple recursive line figures first introduced by R. Dawkins are used to discuss evolutionary principles. We exemplify the creativity of interactive, open-ended evolution.

2 Evolutionary Algorithms for Optimization

In this section, we give a short introduction to adaptive systems, to the preconditions of evolution, and to evolution as adaptive systems. Finally, a basic scheme for evolutionary algorithms is presented.

2.1 Adaptive Systems and Evolution

A general scheme for adaptive systems as proposed by John Holland is presented here [Holland, 1975], [Holland, 1992].

2.2 Preconditions for Evolution to Occur

What are the necessary ingredients for a system to exhibit evolutionary effects?

2.3 Evolution as a Reproductive Plan

A "reproductive plan" is a simplified model of an adaptive system which may serve as the basic scheme of evolutionary algorithms.

3 GA: Genetic Algorithms

In this chapter, we will explore a major class of evolutionary algorithms: Genetic Algorithms (GAs). We will explore various kinds of GA ***chromosome structures***, their main ***genetic operators***, the basic ***evolution scheme***, and look at some ***GA-evolution experiments*** in the domain of parameter optimization.

This chapter on Genetic Algorithms concludes with a brief discussion of GA-building blocks, the Schema Theorem, and a few experiments on GA-schemata.

3.1 GA-Introduction

This section gives a brief overview of Genetic Algorithms, their origins, their primary characteristics with regard to chromosomes, genetic operators, and selection schemes.

3.2 Polyploid GA-Chromosomes

Nature encodes its cellular "programs", which implicitly describe all necessary ingredients for an organism′s development, by a four-letter alphabet known as *nucleotide bases*. In the following sections, GA-chromosomes are defined in a more general form of strings or vectors over a discrete alphabet.

□ Haploid Chromosomes

A single-stranded chromosome of length n is defined as a vector

$$s = (s_1, \ldots, s_n) \text{ with } s_i \in A,\ 1 \leq i \leq n, \qquad (1)$$

for a discrete k-letter alphabet $A = \{a_1, \ldots, a_k\}$.

Part 4: **Finally, a Plus-GA-Strategy again**

3.9 Schemata

So-called "schemata" are used to describe common patterns of genotypic structures that serve as building blocks for the genepool and hence, for the search process.

▫ **Patterns describing genotypic building blocks**

Allele patterns are described by schemata.

▫ **Schematheorem**

The schematheorem describes the effects of selection, mutation, and recombination on the schemata. The schematheorem gives a simple characterization of successful building blocks that will be reproduced exponentially during GA evolution runs.

▫ **Schemata Experiments**

This section presents a few rudimentary examples of schemata and gives an indication of some of the difficulties arising with the computation of schema fitnesses.

4 ES: Evolution Strategies

In this chapter we will explore another major school of evolutionary algorithms: Evolution Strategies (ES).

4.1 ES-Introduction

This section gives a brief introduction to Evolution Strategies, their origins, their primary characteristics with regard to "chromosomes", "genetic operators", selection schemes, and preferred application areas.

4.2 ES-Chromosomes

Here we explain the data structures we will use to represent ES-chromosomes in *Mathematica* and how we implement them. Basically, an *ES-chromosome* $\vec{g}$ is defined as a two-fold vector of the form:

$$\vec{g} = (\vec{p}, \vec{s}) = ((p_1, p_2, \ldots, p_n), (s_1, s_2, \ldots, s_n)) \text{) with } p_i, s_i \in \mathbb{R},$$

where $\vec{p}$ and $\vec{s}$ represent the *object* and *strategy parameters*, respectively.

4.3 ES-Mutations

Mutation is considered the major ES-operator for variations on the chromosomes.
Mutations of the object and strategy parameters are accomplished in different ways. Basically, ES-mutation on a chromosome $\vec{g} = (\vec{p}, \vec{s})$ can be described as follows:

$$\vec{g}_{\text{mut}} = (\vec{p} + N_0(\vec{s}), \alpha(\vec{s})).$$

Here N_0 denotes normal distribution with mean zero, and α defines a function for adapting the strategy parameters.

ES-mutations and selection: The basic idea

What are the main characteristics of the ES-mutation-selection scheme?

Mutating object parameters

How does mutation as performed on object parameters work? What are the most commonly used mutations?

Stepsize adaptation

How can the variances that control mutation step sizes be tuned to better evolutionary performance?

4.4 ES-Recombinations

Introduction

Recombination operators compose new chromosomes from corresponding parts of two or more chromosomes, thus mimicking natural recombination mechanisms as observed in cellular genomes. For the binary case where two ES-chromosomes, $\vec{a} = (\vec{p}_a, \vec{s}_a)$ and $\vec{b} = (\vec{p}_b, \vec{s}_b)$ are to be recombined by an operator ω_{rec}, we can describe the composition of a new chromosome ω' as follows a':

$$\omega' = \omega_{\text{rec}}\left(\vec{a}, \vec{b}\right) = (\vec{p}', \vec{s}') = ((p_1', \ldots, p_n'), (s_1', \ldots, s_n')).$$

Each element of the object and strategy parameter vectors is a recombination of the respective entries of $\vec{a}$ and $\vec{b}$:

$$p_i' = \rho_p(p_{\text{ai}}, p_{\text{bi}}) \text{ and } s_i' = \rho_s(s_{\text{ai}}, s_{\text{bi}})).$$

Here the functions ρ_p and ρ_s define the component-wise recombinations for the object and strategy parameters respectively.

The following sections about ES-recombinations will explore these functions for *discrete* and *intermediate* recombinations as well as different

5 Programming by Evolution

There are no notebooks available for Chapter 5 in the book..

6 EP: Evolutionary Programming

6.1 Evolution of Finite State Automata

This notebook shows an example evolution of finite state automata, which perform prediction tasks.

7 GP: Genetic Programming

In this chapter we explore the "automatic programming" of computers by evolution. We start with a brief history of evolutionary programming by presenting approaches of how to use evolution mechanisms to "breed" computer programs. The genetic programming approach, where terms or simple symbolic expressions are used to encode data structures, is discussed in more detail and demonstrated by example of evolving balanced mobiles. Finally, further variants of genetic programming are outlined.

7.1 GP with Tree Genomes: Introduction

A very successful approach encoding data structures as symbolic expressions was proposed by John Koza [Koza, 1992].

7.2 Terms as Genotypic Structures

Using terms or simple symbolic expressions to represent "program chromosomes" for encoding instructions as well as data structures leads to a flexible and elegant approach for programming by evolution.

7.3 Recombination on Terms

Recombination on terms is performed by interchanging randomly chosen subtrees (subexpressions). This GP term crossover is similar to the GA recombination operators of crossover.

7.4 Mutation on Terms

Mutations on terms substitute a randomly chosen subtree by a newly generated expression.

7.5 GP-Evolutionscheme

The GP evolution scheme is closely related to the GA evolution scheme. The major differences lie in the probabilistic selection of the genetic operators.

7.6 GP in Action: Evolving Balanced Mobiles

The evolution of symbolic expressions that encode mobile structures illustrates the GP evolution scheme and its operators.

- ☐ **Mobiles: Encoding, visualization, and evaluation**
- ☐ **GP-evolution of balanced mobiles**

8 *Evolvica* – Evolution with Symbolic Expressions

This chapter describes the genetic programming system *Evolvica* and its concepts using symbolic expressions and template- or pattern-based genetic operators.

8.1 Templates defining Building Blocks

Similar to the schemata of genetic algorithms, templates are used in *Evolvica's* genetic programming approach for two purposes: (1) for generating expressions and (2) for identifying substructures. In this section we describe how to use templates for expression generation from problem-specific building blocks.

◻ Generating Expressions over Templates

Templates provide the basic building blocks from which expressions are composed.

◻ Expression Generation with Weighted Templates

Attributing the templates with weights implements a competition among building blocks.

8.2 Templates as Filters – Extraction of Contextsensitive Substructures

Templates can be used as filters to identify substructures that obey certain conditions. This is useful for genetic operators such as recombination, where one has to be careful to select "matching" substructures that can be interchanged. This section demonstrates some variants of substructure identification.

◻ Templates identifying substructures

Here we use straight-forward templates to extract expressions.

Hierarchical extraction of substructures

The templates introduced here allow even hierarchical expression filtering.

Embedded structures within structures

Extended templates have to be used to be able to flexibly identify structures embedded within structures.

8.3 Selective Genetic Operators

The following genetic operators, which all work on general symbolic expressions, use templates to identify substructures which are to be modified by the operators.

Mutation

This is an extension of the standard GP mutation operator to work on general symbolic expressions; it uses templates for substructure selection as well as for expression generation.

Recombination

This is an extension of the standard GP recombination operator to work on general symbolic expressions.

Duplication

This is an analogous extension of the standard GA duplication operator to work on general symbolic expressions.

Deletion

This is an analogous extension of the standard GA deletion operator to work on general symbolic expressions.

◻ Permutation and Inversion

This is an analogous extension of the standard GA permutation operator to work on general symbolic expressions.

◻ Encapsulation and Decapsulation

Encapsulation is a genetic operator for contracting and conserving subexpressions by hiding them from other genetic operators.

◻ Pattern Extraction

Pattern extraction is a generalization of the encapsulation operator for different levels of contraction.

8.4 Evolving *AntTrackers*

The usefulness of the genetic operators described above is illustrated by an example of evolving simple programs controlling a robot in a maze, which has to collect food pieces.

◻ Experiment Setup

Here we describe the program encoding, the genetic operators used for this experiment, the population size, etc.

◻ Analysis of an evolution experiment

Analyzing a typical evolution run reveals the dynamics of genetic programming for this optimization task.

9 Computer Models of Developmental Programs

In this chapter we discuss cellular automata (CA) and the evolution of CA rules by means of genetic programming. Lindenmayer-systems are introduced as an alternative way of describing structure formation processes. We also demonstrate how genetic programming can be used to evolve and design L-systems that exhibit certain characteristics.

9.1 Cellular Automata

Cellular automata (CA) are a formidable representative of decentralized computing with rewrite systems. CAs work on a 1-, 2- or 3-dimensional grid of cells, where each cell updates its state according to simple rules which only take the states of nearby cells into account. Usually, cell updates are performed synchronously but independent from each other without global control.

□ Pattern formation in one dimension

Also the simplest cellular automata – a single line of cells – can exhibit a great variety of patterns, as is shown by example of a binary CA.

□ Morphogenesis and selfreproduction

Self-reproduction properties are illustrated through the example of Christopher Langton´s loop automata.

□ Mutation and selfreproduction

What happens if we mutate a set of carefully designed CA rules for a selfreproducing loop?

9.2 Lindenmayer-Systems

10 Evolutionary Inference of L-Systems

10.1 Evolution of Fractal Structures

◻ Evolving 2D Fractal Structures

An example of evolving fractal structures that are encoded by parametrized Lindenmayer systems and turtle interpretation.

Setting the EVOLVICA Directory ...

In case you want to try out the implemented **Evolvica** functions, some of the **Evolvica** notebooks require the definition of the directory where THIS NOTEBOOK resides in.

Please enter the directory path of this notebook (IEC-Evolvica.nb) here:

Below are examples of how to set the directory path under different operating systems:

For Macintosh or Unix Systems:

```
EvolvicaDirectory =
 ToFileName[{"", "Cube OS X", "Users", "jacob",
   "Documents", "EVOLvica"}, "EvolvicaNotebooks"]
```

```
Cube OS X:Users:jacob:
  Documents:EVOLvica:EvolvicaNotebooks
```

```
EvolvicaDirectory =
 ToFileName[{"~jacob", "Desktop", "EvolvicaNotebooks"},
  "EvolvicaNotebooks"]
```

```
~jacob/Desktop/EvolvicaNotebooks/EvolvicaNotebooks
```

For Windows Systems:

```
EvolvicaDirectory = ToFileName[
  {"", "C:", "My Documents", "Summer CPSC", "Evolvica",
   "EvolvicaNotebooks"}, "EvolvicaNotebooks"]
```

```
C:\My Documents\Summer CPSC\
  Evolvica\EvolvicaNotebooks\EvolvicaNotebooks
```

While the cursor is still in the above cell, evaluate the cell by hitting SHIFT-RETURN or select *Kernel* → *Evaluation* → *Evalute Cells* from the menu.

Do the same with the following cell. If you do not get an error message (such as "Cannot set current directory ...") your **Evolvica** directory has been set successfully.

```
SetDirectory[EvolvicaDirectory]
```

```
/Users/jacob/Desktop/
  EvolvicaNotebooks/EvolvicaNotebooks
```

Note: It is assumed that you did not change the recommended **Evolvica** directory structure, i.e., you haven't altered the directory hierarchy of the notebooks you downloaded from the IEC website.