Application of Genetic Algorithms with more Populations for Lindenmayer Systems

Gabriella Kókai[1], *Zoltán Tóth*[2] *and Róbert Ványi*[2]
[1] Institute of Informatics, József Attila University
Árpád tér 2, H-6720 Szeged, Hungary
e-mail: kokai@sol.cc.u-szeged.hu
now visiting Department of Computer Science, Programming Languages
Friedrich-Alexander University of Erlangen-Nürnberg
Martensstr. 3. D-91058 Erlangen, Germany
e-mail: kokai@informatik.uni-erlangen.de
[2] students of the Institute of Informatics, József Attila University
Árpád tér 2, H-6720 Szeged, Hungary
e-mail: h531714,h531774@stud.u-szeged.hu

Keywords: pattern recognition, genetic algorithms, fractal images, Lindenmayer-systems

ABSTRACT

The paper describes a solution of the inverse problem for Lindenmayer systems with genetic programming. Inverse problem means that we try to evolve Lindenmayer grammars to describe fractal images. Genetic algorithm is used to evolve the rewriting rules of the system. As a starting point the solutions of Koza [7] and Jacob [6] is applied but these solutions were extended: for different types of fractal images, and we executed some modifications: more populations are processed parallel during the evolution process, where the selection of the individuals after the meeting of the populations can happen on the basis of either the best or the median fitness value or the user himself can select from the individuals. A further improvement we applied is an adaptation scheme for the application probability of genetic operators. With the help of this method, operators which produce better individuals can be used more frequently, so better individuals are more likely produced. Thus, with the help of them the speed of convergence can be significantly increased.

1 Introduction

This paper describes the solution of the inverse problem for Lindenmayer systems using genetic programming. Inverse problem means that we try to evolve Lindenmayer grammars to describe fractal images.

Let's take the following problem: curve contains only straight line segments is given. The program developed by us in order to solve the problem allows the user to select different types of fractal images but in this paper we focus only on the so-called quadratic Koch island (see Figure 1) to demonstrate how our system works. The goal is to define this curve with the help of some kind of grammar. The Lindenmayer grammars seemed to be a suitable descriptive grammar.

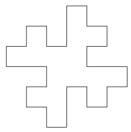

Figure 1: The Koch island

The Lindenmayer systems (L-systems) are grammatical rewriting systems. They contain an initial axiom and one or more rules. With the help of such a grammar complex objects can be described starting from the axiom and using the productions [11].

When choosing the representation form, we took into consideration that L-systems can be evolved using genetic algorithms ([6]). The genetic algorithms can be briefly described as follows: a model of natural evolution ([1]) is often used as a way of solving optimization problems ([5],[12],[13]). These algorithms are based on the collective adaptation and learning ability of *individuals*. Individuals form a *population*, and each individual represents a possible solution of the problem. At the beginning, the individuals in the population are initialized randomly, and then they can be modified by the operators *selection*, *mutation* and *recombination*. These modifications lead to the evolution of the individuals; during the process better and better individuals appear. Thus, in later generations individuals that represent better solutions more likely appear.

In this paper an improved version of the solution of the problem introduced above (firstly presented by John R. Koza [7] and refined among others by Jacob [6]) is described. Some modifications of the running parameters of the genetic algorithm used by Koza are made, and some changes of the representation way of individuals are executed. With the help of these modifications the speed of convergence can be significantly increased. But the main ideas are that we applied an extended version of genetic algorithms: we processed more populations parallel (the number of populations also can be determined by the user), and after a while we made the populations meet. Thus, populations could change each other's individuals. It can happen on the basis of either the best or median fitness value or by user intersection, which means that the versatility of populations could be increased.

A further improvement we applied is an adaptation scheme for the application probability of genetic operators. With the help of this method, operators which produce better individuals can be used more frequently, so better individuals are more likely produced. This method also increases the speed of convergence.

The whole system *ELSFI* (Evolving Lindenmayer Systems for Fractal Images) which solves the problem is implemented in Java language. To draw the individuals, the well-known turtle graphics system is applied.

We proceed as follows. First in Section 2 a brief overview of the theory is presented. In Section 3 we show how the rules describing the quadratic Koch island can be evolved and give some details about the program *ELSFI*. The comparison with Koza's work is presented in Section 4 and finally Section 5 contains a summary and some comments on future work.

2 Fundamentals of the evolution using L-systems

In this section we give a short overview of concepts used in this paper. First we define the Lindenmayer system. The turtle graphic draws the individuals generated by the grammar on the screen. Then a short informal description of genetic algorithms used to evolve the grammar rules is presented. The rest of this section contains an introduction to the fractal images, and we describe a special kind of these images, the so-called quadratic Koch island, applied to demonstrate the work of the genetic algorithm.

2.1 Lindenmayer systems

Lindenmayer introduced kinds of descriptive grammars in order to describe natural development processes ([8]). These are called Lindenmayer grammars. An object described by a Lindenmayer grammar is called Lindenmayer system (L-system). The definition of a Lindenmayer system is the following: $G = (\Sigma, \pi, \alpha)$, where

- $\Sigma = \{s_1, s_2, ..., s_n\}$ is an alphabet
- $\pi : \Sigma^+ \to \Sigma^*$ is a homomorphism
- $\alpha \in \Sigma^+$ is an axiom.

Thus, a Lindenmayer system can be characterized as the following:

- Description the initial state
- One or several rewriting rules: these determine how the object is developed
- Description of the current state: shows the attributes of the simulated object. This description is a series of state indicators created by modifications of the initial state. These modifications have to correspond to the rewriting rules. Depending on how many times the rewriting rules are applied, we can speak about *generation* $0, 1, \ldots, n$. Generation 0 is the initial state

The L-systems can be classified as follows:

Context-free L-systems
An L-system is context-free if its state indicators do not affect the evolution of the state indicators neighbouring them. This is called $0L$*-system*. The 0 shows that the evolution of a state indicator depends only on the state indicators in distance 0 from it. Later similar tokens $(1L, 2L, (m, n)L)$ are used to mark context-sensitive L-systems. The simplest L-systems are the deterministic $0L$-systems ($D0L$-systems). In this paper only this kind of L-systems is studied.

D0L-systems
The most important characteristic of $D0L$-systems is that each state indicator has one rewriting rule at

the most. Thus, the evolution of the $D0L$-system is unambiguously determined. The mathematical definition of $D0L$-systems is similar to the one described above. Σ^* denotes the set of all finite sequences generated from the elements of the alphabet. Σ^+ denotes the $\Sigma^* - \lambda$, where λ is the empty sequence. If the rewriting rule for an element $s_i \in \Sigma$ is not determined $\pi(s_i) = s_i$.

The application of rewriting rules can be performed as follows: all sequences can be decomposed to $\varepsilon_1, \varepsilon_2, ..., \varepsilon_n$ parts. Thus $\pi(\varepsilon) = \pi(\varepsilon_1 \varepsilon_2 ... \varepsilon_n)$ and if $\varepsilon_i = s_j$, then $\pi(\varepsilon_i) = \pi(s_j)$, otherwise ε_i can be broken into parts of sequences too.

Other types of L-systems are the context-sensitive L-systems and the parametric L-systems. The most important characteristic of L-systems compared with Chomsky grammars is *parallel rewriting*. When the current string is rewritten, all the occurrences of each symbol staying on an left side of a rewriting rule are simultaneously replaced by appropriate the right side of the rule.

2.2 Turtle

The turtle can visualize complex objects described in a string format. For example, the string generated by the L-system for the Koch-island can have the following form: $FRFLFLFFRFRFLF$ over the alphabet $\Sigma = \{F, L, R, [,]\}$. The symbol F might be interpreted as a straight line segment and the adequate turtle interpretation is one unit forward. Operation L means a rotation of the turtle in the left direction and operation R moves the turtle in the opposite direction. The brackets represent the turtle's stack-utility which means that after the execution of the bracketed segment the turtle goes back to its position and orientation having by the start of the evaluation of the bracket. The definition of the turtle is well-known so it is not described in this paper in detail.

2.3 Evolution algorithms

Evolution algorithms are computer-based problem solving systems which use evolution processes as they can be observed in nature ([1]). Main types of evolution algorithms are evolution strategies ([12], [13]), genetic algorithms ([4], [5]) and genetic programming ([7]). All of them are based on the simulation of evolution of individuals by means of natural *selection*, *mutation* and *reproduction*.

More exactly, evolution algorithms work with *populations of structures* which are evolved by using of *selection*, *mutation* and *reproduction*. We provide each individual with a *fitness value*, which shows how the individual solves the problem. Reproduction selects the individuals with the high fitness values, and recombination and mutation work on these individuals. Thus, the preservation of good attributes is assured.

Solving the problem of finding rewriting rules for the description of Koch island genetic algorithm is used, so here follows a brief overview of one kind of genetic algorithm, the *genetic programming. A more detailed description can be found in [7].*

2.3.1 Genetic programming

A genetic algorithm is a model of a machine which performs operations of natural evolution handling a population whose *individuals are represented by* chromosomes. Genetic programming uses unspecified length of chromosomes. It means that the size of the chromosomes can be dynamically changed during the process.

It can be noticed that evolution in nature is not a guided process. In natural processes individuals race for environmental resources. Some are better than the others. The better ones are more likely survive, so they can reproduce their good attributes to their descendants.

In molecular level the pairs of chromosomes collide, exchange their parts, and then leave each other. This operation is called *recombination*. The individuals which recombine during genetic algorithms are selected on the basis of their fitness value.

Mutation also operates in the algorithm, but its importance is a subject of discussions. Anyway, it can not be too strong, because during the genetic algorithms we search for the solution not in a stochastic way.

When implementing the basic genetic programming, generally the following loop is used:

- The fitness value of each individual in the population is determined
- A new population is generated by applying genetic operators recombination, mutation and fitness-proportional selection
- Consider the new population as the actual population and continue the process with the next iteration

One iteration of the process described above is called *generation*. The first (0.) generation works with a population of randomly created individuals.

After that, genetic operators work according to the individuals' fitness values, and so the individuals show better and better behaviour.

In all cases it must be declared how the fitness value of each individual in the population can be calculated. In our demonstrative example the individuals are made up of straight lines, so we computed the fitness value according to that how many similar and how many different lines the individual (I) has got with the quadratic Koch island (K). Formally the fitness $f(x) = 9|M| - |L|$ where $M = K \cap I$ is the lines which are similar and $L = K \backslash I$ is the different segments. The used constants 9 and 1 are established by experiences.

2.3.2 *Adaptation of the weight of the genetic operators*

As we mentioned before the descendants during the process are created by recombination, mutation and reproduction. The weight of each operator is adapted by observing their effectiveness: during one step (when generating a population of descendants) the effectiveness (the individuals generated with this operator have better fitness value than their parents) of each operator is observed. The more effective the operator is, the more likely we are to use it in the next step. Thus, the weight of the operators is continually changed during the evolution process.

The probability of operator $\omega \in \Omega$ in generation t is

$$\mathcal{P}(\omega) = \frac{\gamma(\omega, t)}{\sum\limits_{\omega' \in \Omega} \gamma(\omega', t)}$$

where Ω is the set of operators, and $\gamma(\omega, t)$ is the weight of operator ω computed by

$$\gamma(\omega, t) = \gamma(\omega, t-1) \cdot \Delta\gamma(\omega, t) \cdot \gamma_{decay}(\omega).$$

The operator-specific constant $0 \leq \gamma_{decay}(\omega) \leq 1$ is introduced in order to prevent an operator from generating too many individuals in a generation. That is, no matter how bad an operator is, it still can produce some individuals (we must not exclude any operators from the process). $\Delta\gamma(\omega, t)$ shows how the operator ω increased the average fitness of the population in the last step. Its calculation depends on the number of operands used in the operator. For operators with one operand, the method is the following:

$$\Delta\gamma(\omega, t) = |P(\omega, t)|^{-1} \cdot \sum_{s \in P(\omega, t)} \frac{\phi(s)}{\phi(Parent(s))},$$

where $P(\omega, t)$ is the set of individuals created by operator ω in generation t, and $\phi(s)$ is the fitness value of the individual $s \in P(\omega, t)$. $Parent(s)$ is the parent of the individual $s \in P(\omega, t)$.

The method of computing the $\Delta\gamma(\omega, t)$ value for operators with two operands (e.g. recombination) is the following:

$$\Delta\gamma(\omega, t) = \frac{1}{2} \cdot |P(\omega, t)|^{-1} \cdot \sum_{s \in P(\omega, t)} \phi^*(s),$$

where

$$\phi^*(s) = \max\left\{\frac{\phi(s)}{\phi(parent_1(s))}, \frac{\phi(brother(s))}{\phi(parent_2(s))}\right\}.$$

More details can be found in [6].

2.4 Fractals

In the last few years Benoit Mandelbrot [9] noticed that the world is not as simple as the ancient philosophers had thought. In the ancient space approach, there are certain plane figures and geometric solids such as point, line, rectangle, cube or orb. Ancient philosophers thought that every object in the world could be described as a combination of these figures and solids. But if we examine real objects, we must notice that there are no perfect points, lines or cubes.

Mandelbrot pointed out that galaxies, plants, snowflakes, mountains are all examples of so-called fractals, dashed structures, dashed-dimensignal structures. More about fractals can be found in [3] and [10] .

In this paper the quadratic Koch island is selected as an illustrative example. The axiom describing it with L-system is $FLFLFLF$ and there is only one rewriting rule: $F \to FLFRFRFFLFLFRF$. The form of the curve defined by the first iteration for the Koch island can be seen in Figure 1. Some other shapes can be evolved by the program are shown in the Figure 2.

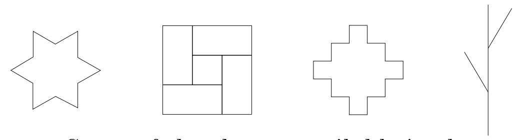

Figure 2: Some of the shapes available in the program ELSFI: The von Koch snowflake, one kind of the Koch island, approximation of Sierpinski and a tree

3 The inverse problem of L-systems

In this section we describe our version to solve the inverse problem of L-systems. We demonstrate how the foregoing solutions ([6], [7]) can be improved and give some details about the using of our program *ELSFI* to the problem of finding a rewriting rule to the quadratic Koch island.

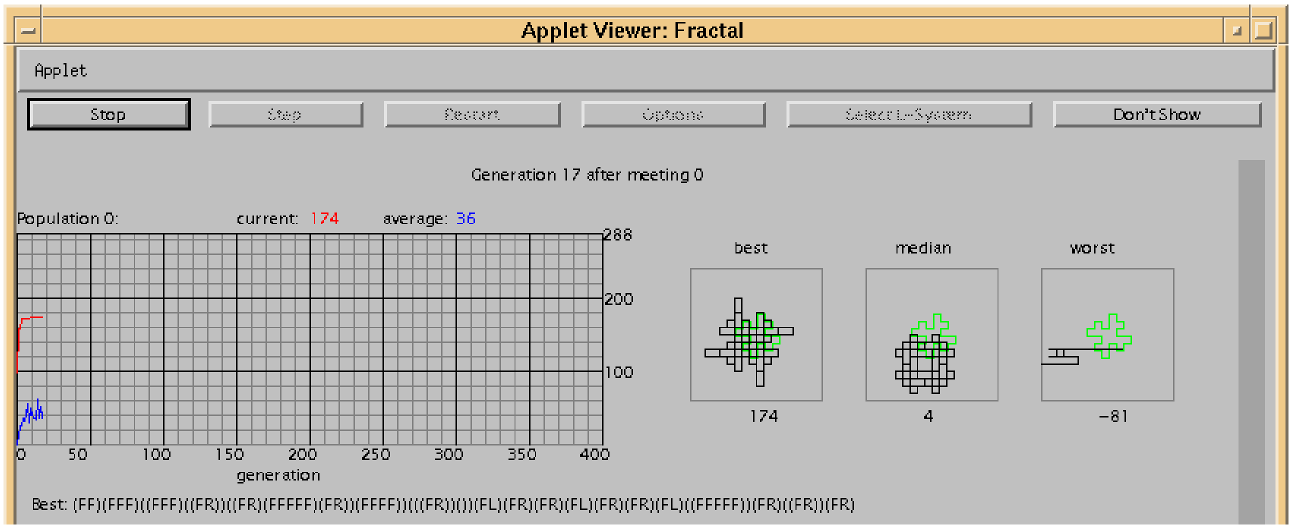

Figure 3: The GUI

We implemented the algorithm which solves the given problem in Java language.

The program *ELSFI* can evolve Lindenmayer-systems for different fractal images. The user can choose the type of the image he wants to evolve through a dialog box. Here we put the most frequently used shapes, and the user can also define a shape by its axiom and rewriting rules. Some of the available shapes can be seen in Figure 2.

The program can be handled by an GUI (see in Figure 3) and runs the genetic algorithm. On the top of the GUI buttons can be found which are responsible for starting or suspending the genetic algorithm, executing in the algorithm step by step, and for setting the genetic parameters (see later). The chart and some pictures for each population are printed. The chart shows how the population has evolved from the beginning of the evolution, and the pictures of the best, the median and the worst individuals of the population can be seen.

By clicking on the fourth button *Options*, the following parameters can be changed by the user: the number of individuals in the population, the number of populations, the generation number after the populations meet, the initial percentage rate of each operator and the strength of mutation applied at the meeting of the populations.

When the populations meet, the genetic parameters are applied on the populations themselves; at this point, the populations are considered as individuals of the population.

The user can also change the probability of each individual to become a parent by using the Weight-input box. In this dialog box the user can draw a curve, thus determining the probability of the individuals to be used as parents.

One point, where our algorithm differs from the algorithm used by John Koza is: he worked with *one* population containing 4000 individuals, used the 7 best individuals as parents, and applied only *recombination* and *reproduction* as genetic operators and he *fixed* this setting of the parameters. Of course the parameters can be optionally changed in our implementation in order to achieve faster convergence.

4 Experimental results

During our work we studied the results of the algorithm implemented by John R. Koza, and compared these results to the results of our modified algorithm. In this section we present the two algorithms and the results of comprehension.

The most important difference between our and John Koza's algorithm is the following: Koza did not determine the possible orders of the turtle commands following each other, we used sequences in rewriting rules, and determined which turtle commands a sequence could contain. This way during the evolution more regular individuals appeared, which means faster convergence.

4.1 Results for the quadratic Koch island

To determine the rewriting rule for iteration 1 of quadratic Koch island, John Koza used a genetic algorithm. The axiom of the Lindenmayer system was the same as our starting point. As we mentioned before the algorithm differs in the parameters of the genetic algorithm: in rewriting rules, as he did not use sequences, such rewriting rule parts evolved which did not draw any lines, as turned the turtle left and right in the same position, or moved it back and forth on a line.

In both experimental cases the fitness value is computed in the same way. With these parameters, John R. Koza got the following results: The fitness value of the *best-of-generation* individual in *generation 0* almost reached half of the fitness value of the ideal individual. But as it can be seen in Figure 4, the *best-of-generation* individual in *generation 1* is almost the same as the best-of-generation individual in *generation 0*. Thus, its fitness value differs only a little from the others fitness value.

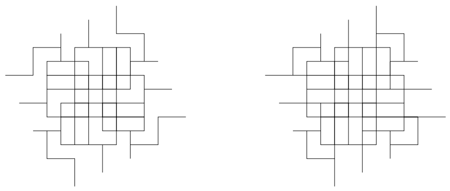

Figure 4: The best-of-generation individuals in *generation 0* and *generation 1* in Koza's algorithm

Koza found an individual in *generation 0* relatively close to the ideal individual. But later convergence slows down because of the unregulated formation of the descendants, and the structural complexity of the new individuals are rather high. Despite this at the end of the algorithm he got a valid result, because too complex individuals died off during the evolution. The algorithm, which worked with populations of *4000* individuals, produced an ideal individual in *generation 50*. In this paper we do not detail John Koza's experimental results, but we must notice that the handling of *4000* individuals required a rather strong hardware background.

4.2 Regulated evolution of Lindenmayer systems

We searched for an algorithm that can produce a solution in relatively short time with smaller population size. With this aim we introduced the idea of sequences in the rewriting rules. The formal definition of a sequence for the quadratic Koch island is the following:

$$SEQ = (FL)$$
$$SEQ = (FR)$$
$$SEQ = (\mid F \mid)$$
$$SEQ = (\mid SEQ \mid)$$

The rewriting rules of Lindenmayer systems are defined as a finite series of sequences. In the recursive definition the turtle commands contained by a sequence are written in round brackets; F, L and R mean the turtle commands, and a sign in |-s means that the sign is repeated finite times. Thus those movements of the turtle with no visible effect are excluded. In addition, other new genetic operators than recombination and selection are applied. With this modifications, the following results are achieved: Note that the fitness value of the ideal individual using the method described in Section 2.3 is 288. Best-of-generation individuals of *generations* 0, 2 and 3 of *population 1* can be seen in Figure 5.

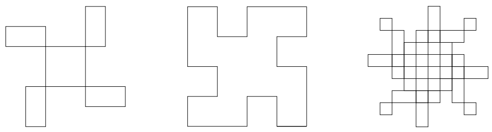

Figure 5: Best-of-generation individuals in *generation 0, 2* and *3*

The fitness value of the best-of-generation individual in *generation 0* was 128. After applying the genetic operators on the population, in *generation 2*, the fitness value of the best-of-generation individual was 176.
As it can be seen, due to our modifications in the algorithm, the velocity of convergence has grown. The fact that the best-of-generation individual in *generation 3* had a fitness value of 208 supports this statement, too.

Since several running parameters of the genetic algorithm can be modified by the user, here we show the results of running our solution on some different parameters:

number of populations	number of ind.'s per population	parent selection	best fitness in gen. #0	almost ideal ind. appeared in gen. #
4	100	linear	128	33
4	35	best	88	49
8	35	linear	76	88

Table 1: Comprehension of the running parameters

As it can be seen, more individuals mean faster convergence. The same can be observed about the number of populations.

5 Summary, future work

In this paper it is showed how the first iteration of quadratic Koch island can be described with Lindenmayer systems. We used genetic algorithm to evolve the rewriting rule of the deterministic Lindenmayer system. The foregoing solutions ([6],[7]) are improved in the way that more populations are used and the weight of the operators applied in the evolution is adapted. During realization, we implemented the genetic algorithm and the Lindenmayer system in such a way that further modifications and development can be executed easily, and the newly created system can have the effectiveness shown above.

We plan to extend our system to manipulate 3-dimensional objects. In this paper only $D0L$-systems were used. We are planning to make experiments using *context-sensitive* L-systems. A character recognition program which uses genetic algorithms could be an interesting application. Fractal based picture condensing can be mentioned as an other practical application. Both are based on finding a similar shape to a given one, and one describing it by some kind of grammars.

References

[1] Darwin, C.: *On the Origin of Species* Murray, London, 1859.

[2] Davis, L.: *Adapting Operator Probabilities in Genetic Algorithms* Morgan Kaufmann, 1989.

[3] Deking, F. M.: *Recurrent Sets: A Fractal Formalism* In: Report 82-32, Delft University of Technology, 1982.

[4] Goldberg, D.E.: *The Genetic Algorithm Approach Why, How and What Next* In: Narenda 1986

[5] Holland, J. H.: *Adaptation of Natural and Artificial Systems* Univ. of Michigan Press, Ann Arbor, Mich., 1975.

[6] Jacob, C.: *Principia Evolvica - Simulierte Evolution mit Mathematica* Dpunkt Verlag, 1997.

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

[8] Lindenmayer, A.:*Mathematical models for cellular interaction in development* In: Journal of Theoretical Biology 18: 280-315 1968.

[9] Mandelbrot, B. B.:*The fractal geometry of the nature* W.H. Fruman, San Francisco, 1982.

[10] Peitigen, H.-O., Jürgens, H., Saupe D.:*Chaos, Bausteine der Ordnung* Klett-Cotta/Springer Verlag New York 1994

[11] Prusinkiewitz, P., Lindenmayer, A.:*The Algorithmic Beauty of Plants* Springer-Verlag, New York, 1990.

[12] Rechenberg, I.: *Evolutionsstrategie '94, Werkstatt Bionik und Evolutionstechnik* Band 1, Fromman-Holzboog, Stuttgart, 1973.

[13] Schwefel, H.-P.: *Numerische Optimierung von Computer-Modellen mittels der Evolutonsstrategie* In: Interdisciplinary Systems research (26), Birkhäuser, Basel, 1977