Chapter 11
Chaos Synthesis by Evolutionary Algorithms

Ivan Zelinka, Guanrong Chen, and Sergej Celikovsky

Abstract. This chapter introduces the notion of chaos synthesis by means of evolutionary algorithms and develops a new method for chaotic systems synthesis. This method is similar to genetic programming and grammatical evolution and is applied alongside evolutionary algorithms: differential evolution, self-organizing migrating, genetic algorithm, simulated annealing and evolutionary strategies. The aim of this investigation is to synthesize new and "simple" chaotic systems based on some elements contained in a pre-chosen existing chaotic system and a properly defined cost function. The investigation consists of two case studies based on the aforementioned evolutionary algorithms in various versions. For all algorithms, 100 simulations of chaos synthesis were repeated and then averaged to guarantee the reliability and robustness of the proposed method. The most significant results are carefully selected, visualized and commented in this chapter.

Ivan Zelinka
Tomas Bata University in Zlin, Faculty of Applied Informatics, Nad Stranemi 4511,
Zlin 76001, Czech Republic
and
VSB-TUO, Faculty of Electrical Engineering and Computer Science, 17. listopadu 15,
708 33 Ostrava-Poruba, Czech Republic
e-mail: zelinka@fai.utb.cz

Guanrong Chen
Department of Electronic Engineering, City University of Hong Kong, Kowloon,
Hong Kong SAR, P.R. China
e-mail: eegchen@cityu.edu.hk

Sergej Celikovsky
Institute of Information Theory and Automation,
Academy of Sciences of the Czech Republic, Faculty of Electrical Engineering,
Czech Technical University in Prague
e-mail: celikovs@utia.cas.cz

11.1 Introduction

Deterministic chaos, discovered by E. Lorenz [26] is a fairly active area of research in the last few decades. The Lorenz system produces a well-known chaotic attractor in a simple three-dimensional autonomous system of ordinary differential equations [26], [41]. For discrete chaos, there is another famous chaotic system, called logistic equation [27], which was found based on a predator-prey model showing complex dynamical behaviors. These simple models are widely used in the study of chaos today, while other similar models exist (e.g., canonical logistic equation [14] and 1D or 2D coupled map lattices [40]). To date, a large set of nonlinear systems that can produce chaotic behaviors have been observed and analyzed. Chaotic systems thus have become a vitally important part of science and engineering at the theoretical as well as practical levels of research. The most interesting and applicable notions are, for example, chaos control and chaos synchronization related to secure communications, among others.

Recently, the study of chaos is focused not only along the traditional trends but also on the understanding and analyzing principles, with the new intention of controlling and utilizing chaos toward real-world applications as demonstrated in [6], [45] and many references therein. The term chaos control was first used by Ott, Grebogi and Yorke in 1990. It represents a process in which a control law is derived and used such that the original chaotic behavior can be stabilized on a constant level of output value or a periodic cycle. Since the first experimental report on chaos control, many control methods have been developed and some are based on the first approach [33], including pole placement [15], [56] and delay feedback [36], [20], [21], to name just a couple. Many methods were adapted to spatiotemporal chaos represented by Coupled Map Lattices (CML). Control laws derived for CML are usually based on existing system structures [40], or using an external observer [5], etc. Evolutionary approach for control was also successfully developed in, for example, [38], [37], [50].

Simultaneously with these research activities, some new chaotic systems were found, like the chaotic Chen system [7], [42], which according to [44], [43] is a dual system of the Lorenz system. Other well-known chaotic systems were discovered, including dissipative ones like Lozi, Tinkerbell, Ikeda, Sinai, Burger, Duffing systems, and conservative ones like Chirikov, Arnold, Baker, Nose-Hoover and Henon-Heiles systems. These chaotic systems were mostly derived from some known physical systems. A typical example is the logistic equation obtained based on the predator-prey system or the Lorenz system derived from an atmospheric model. On the contrary, another direction of research was evolving about chaotic systems synthesis. As a few representative examples, [55] investigated an algorithm for computing heteroclinic orbits with possible use in chaos synthesis. This investigation partly used ideas on chaos synthesis and synchronization from [1]. In [12] there was another investigation, which is based mostly on hardware to generate multiple-scroll strange attractors. Basically very similar research was also carried out in [11], [10].

Methods used in generating new chaotic systems from physical systems or from "manipulations" (e.g., control and parameter estimation [40], [18]) are based on deterministic mathematical analysis. Along with these classical methods, there are also numerical methods based partly on deterministic and partly on stochastic methods, called evolutionary algorithms (EAs) [2]. Evolutionary algorithms were used in searching solutions in many computationally hard problems including classes of P and NP problems [13]. In chaos studies, they have been used for chaos control [49], [50], [37], among others.

The aim of this chapter is to show that EA-based symbolic regression (i.e., handling with symbolic objects to create more complex structures) is capable of synthesizing chaotic behavior in the sense that the mathematical descriptions of chaotic systems are synthesized symbolically by means of evolutionary algorithms. The ability of EAs to successfully solve this kind of black-box problems has been proven (see, for example, [51], [28]), and is reinforced once again here in this chapter.

The paper is organized as follows. The first part outlines the motivation of the research. This is followed by a brief survey of evolutionary algorithms, along with a brief description of symbolic regression methods used with evolutionary algorithms. Next, the method of symbolic regression, called analytic programming, is described in more detail, which will be used in the rest experiments. Evolutionary synthesis of chaos is then studied, and finally experimental results are reported, followed by the conclusion.

11.2 Motivation

In recent years, interests in soft computing methods are increasing, including in particular evolutionary algorithms. These algorithms are based on similar principles of biological evolution in the real world. The aim of EAs is to solve computationally hard problems which are too complex to be solved by conventional methods. In its canonical form, EAs can be used only for numerical estimation of parameters (usually, arguments of a given cost function). Together with EAs in the canonical form, another modification allows to use EAs as a symbolic "constructors", i.e., a processor, for synthesizing complex structures in a symbolic way, based on some predefined simple elements (mathematical operators or electronic elements like diode, transistor, etc.). The term "symbolic way" stipulates that mathematical structures and equations, electronic systems, etc., are generated from those simple elements just mentioned.

Given the above background, the main motivation of this research was the question *"Is it possible to synthesize the mathematical description of a new chaotic system, based on simple and elementary mathematical objects, by means of evolutionary computation?"* This question was also based partially on the fact that in engineering applications, it is very often vitally important to know not only when chaos can be generated but also how to generate it [6], [34]. This is extremely important in cryptography, for example, where chaotic systems are often used in the design. From a mathematical point of view, it is quite clear that there are some classes of chaotic systems which can be represented by one canonical form (one class – one canonical form) [14]. However, generally speaking, it is not so easy to exactly synthesize a chaotic system with specified features by means of classical mathematical methods. A positive answer to the question mentioned above would open possibilities to synthesize not only a set of not-yet-described chaotic systems, but also some chaotic systems with predefined features. It is believed that such possibilities would have an important impact on engineering design of various complex nonlinear systems, especially chaotic systems.

11.3 Brief Review of the Selected Evolutionary Algorithm

For both numerical and symbolic experiments described below, stochastic optimization algorithms such as Differential Evolution (DE) [35], Self Organizing Migrating Algorithm (SOMA) [48], Genetic Algorithms [17], [4] for Simulated Annealing (SA) and [9] or [3] for Evolutionary Strategies (ES) are selected to use. For detail description of selected evolutionary algorithms see Chapter 6.

11.4 Symbolic Regression – An Introduction

The term "symbolic regression" represents a process during which measured data sets are fitted thereby a corresponding mathematical formula is obtained in an analytical way. An output of the symbolic expression could be, for example, $\sqrt[N]{x^2+\frac{y^3}{k}}$, and the like. For a long time, symbolic regression was a domain of human calculations but in the last few decades computers as generally used for symbolic computation.

The initial idea of symbolic regression by means of a computer program was proposed in Genetic Programming (GP) [22], [23]. The other approaches are Grammatical Evolution (GE) developed in [39] and Analytic Programming (AP) in [53]. Other interesting investigations using symbolic regression were carried out in [19] on Artificial Immune Systems and Probabilistic Incremental Program Evolution (PIPE), which generates functional programs from an adaptive probability distribution over all possible programs. As an extension of GE to the another algorithms is also [30], where DE was used with GE. Symbolic regression is schematically depicted in Fig. 11.1. Generally speaking, it is a process which combines, evaluates and creates more complex structures based on some elementary and noncomplex objects, in an evolutionary way. Such elementary objects are usually simple mathematical operators $(+,-,\times,...)$, simple functions (*sin*, *cos*, *And*, *Not*, ...), user-defined functions (simple commands for robots – MoveLeft, TurnRight, ...), etc. An output of symbolic regression is a more complex "object" (formula, function, command,...), solving a given problem like data fitting of the so-called Sextic and Quintic problem described by eq. (11.1) [25], [52], randomly synthesized function by eq. (11.2) [52], Boolean problems of parity and symmetry solution (basically logical circuits

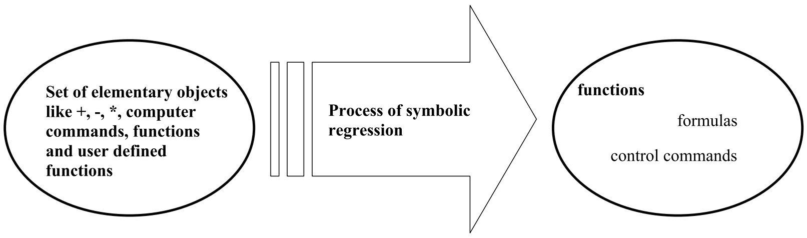

Set of elementary objects like +, -, *, computer commands, functions and user defined functions
Process of symbolic regression
functions
formulas
control commands

Fig. 11.1 Symbolic regression - schematicall view

synthesis) by eq. (11.3) [24], [53], or synthesis of quite complex robot control command by eq. (11.4) [23], [32]. Equations (11.1)–(11.4) mentioned here are just a few samples from numerous repeated experiments done by AP, which are used to demonstrate how complex structures can be produced by symbolic regression in general for different problems.

$$x\left(K_1+\frac{\left(x^2K_3\right)}{K_4\left(K_5+K_6\right)}\right)*\left(-1+K_2+2x\left(-x-K_7\right)\right) \tag{11.1}$$

$$\sqrt{t}\left(\frac{1}{\log(t)}\right)^{\sec^{-1}(1.28)}\log^{\sec^{-1}(1.28)}\left(\sinh\left(\sec\left(\cos(1)\right)\right)\right) \tag{11.2}$$

$$\begin{array}{l} Nor[(Nand[Nand[B||B,B\&\&A],B])\&\&C\&\&A\&\&B,\\ Nor[(!C\&\&B\&\&A||!A\&\&C\&\&B||!C\&\&!B\&\&!A)\&\&\\ (!C\&\&B\&\&A||!A\&\&C\&\&B||!C\&\&!B\&\&!A)||\\ A\&\&(!C\&\&B\&\&A||!A\&\&C\&\&B||!C\&\&!B\&\&!A),\\ (C||!C\&\&B\&\&A||!A\&\&C\&\&B||!C\&\&!B\&\&!A)\&\&A]] \end{array} \tag{11.3}$$

$$\begin{array}{l} \text{Prog2[Prog3[Move, Right, IfFoodAhead[Left, Right]],}\\ \text{IfFoodAhead[IfFoodAhead[Left, Right], Prog2[IfFoodAhead[}\\ \text{IfFoodAhead[IfFoodAhead[Left, Right], Right], Right],}\\ \text{IfFoodAhead[Prog2[Move, Move], Right]]]]} \end{array} \tag{11.4}$$

11.4.1 Genetic Programming

Genetic programming was the first tool for symbolic regression carried out by means of computers instead of humans. The main idea comes from genetic algorithms (GA), which was used in GP [22], [23]. Its ability to solve very difficult problems is well proven; for example, GP performs so well that it can be applied to synthesize highly sophisticated electronic circuits [24].

The main principle of GP is based on GA, which is working with populations of individuals represented in LISP programming language. Individuals in a canonical

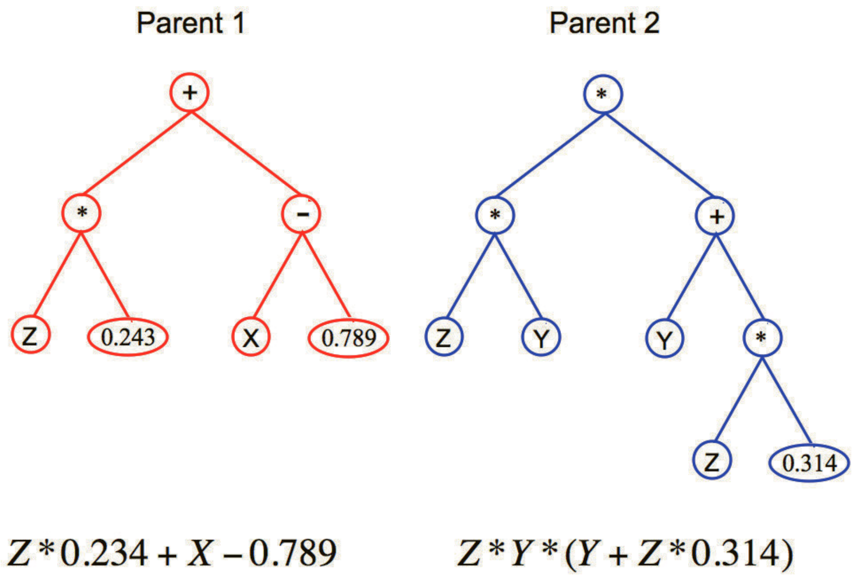


Fig. 11.2 Parental trees

form of GP are not binary strings, different from GA, but consist of LISP symbolic objects (commands, functions, ...), etc. These objects come from LISP, or they are simply user-defined functions. Symbolic objects are usually divided into two classes: functions and terminals. Functions were previously explained and terminals represent a set of independent variables like x, y, and constants like π , 3.56, etc.

The main principle of GP is usually demonstrated by means of the so-called trees (basically graphs with nodes and edges, as shown in Fig. 11.2 and Fig. 11.3, representing individuals in LISP symbolic syntax). Individuals in the shape of a tree, or formula like $0.234Z+X-0.789$, are called programs. Because GP is based on GA, evolutionary steps (mutation, crossover, ...) in GP are in principle the same as GA. As an example, GP can serve two artificial parents – trees on Fig. 11.2 and Fig. 11.3, representing programs $0.234Z+X-0.789$ and $ZY(Y+0.314Z)$. When crossover is applied, for example, subsets of trees are exchanged. Resulting offsprings of this example are shown in Fig. 11.3.

Thereafter, offspring fitness is calculated such that the behavior of the just-synthesized and evaluated individual-tree should be as much as possible similar to the desired behavior. Here, desired behavior can be understood to be like a measured data set from some process (a program that should fit them as well as possible) or like an optimal robot trajectory, i.e., when the program is realizing a sequence of robot commands (TurnLeft, Stop, MoveForward,...) leading as close as possible to the final position. This is basically the same for GE.

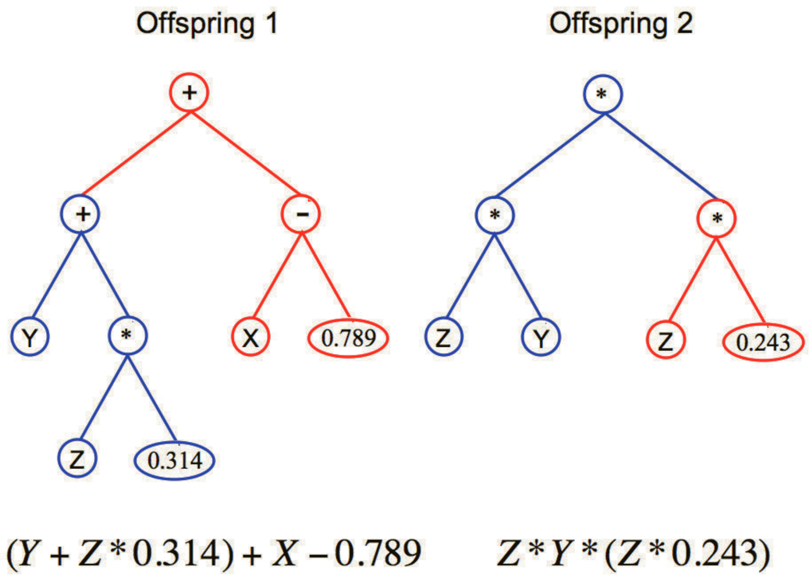


Fig. 11.3 Offsprings

For detailed description of GP, see [23], [25] and for on-line working example, visit [http://evonet.lri.fr/CIRCUS2/node.php?node=56].

11.4.2 Grammatical Evolution

Another method for the same task in view of the resulting program alike GP was developed in [29] called grammatical evolution (GE). GE has one advantage compared to GP and this is the ability to use arbitrary programming languages, not only LISP as in the case of the canonical version of GP. In contrast to other evolutionary algorithms, GE was used only with a few search strategies, and with a binary representation of the populations [29]. Last successful experiment with DE applied on GE was also done in [30]. Grammatical evolution is in its canonical form based on GA, thanks to a few important changes it has in comparison with GP. The main difference is in individual coding.

While GP manipulates in LISP symbolic expressions, GE uses individuals based on a binary strings. These are transformed into integer sequences and then mapped into a final program in the Backus-Naur form (BNF) [29], as explained by the following artificial example. Let $T = \{+, -, \times, /, x, y\}$ be a set of operators and terminals and let F = {epr, op, var} be the so-called nonterminals. In this case, the grammar used for final program synthesis is given in Table 11.1. The rule used for individuals transforming into a program is based on eq. (11.5) below. Grammatical evolution is based on binary chromosome with a variable length, divided into the so-called codons (range of integer values, 0-255), which is then transformed into an integer domain according to Table 11.2.

$$\begin{aligned} &\text{unfolding} = \text{codon mod rules} \\ &\text{where rules is number of rules for given nonterminal} \end{aligned} \qquad (11.5)$$

Table 11.1 Grammatical evolution - rules

Nonterminals		Unfolding	Index
expr	::=	op expr expr	0
		var	1
op	::=	+	0’
		-	1’
		*	2’
		/	3’
var	::	X	0”
		Y	(1”)

Table 11.2 Grammatical evolution - codon

Chromosome	Binary	Integer	BNF index
Codon 1	101000	40	0
Codon 2	11000011	162	2’
Codon 3	1100	67	1
Codon 4	10100010	12	0”
Codon 5	1111101	125	1
Codon 6	11100111	231	1”
Codon 7	10010010	146	Unused
Codon 8	10001011	139	Unused

Synthesis of an actual program is described as the following: start with a nonterminal object expr. Because the integer value of Codon 1 (see Table 11.2) is 40, according to eq. (11.5) one has an unfolding of *expr* = *op expr expr* (40 mod 2, 2 rules for *expr*, i.e., 0 and 1). Consequently, Codon 2 is used for the unfolding of *op* by * (162 mod 4), which is terminal and thus the unfolding for this part of program is closed. Then, it continues in unfolding of the remaining nonterminals (*expr expr*) till the final program is fully closed by terminals. If the program is closed before the end of the chromosome is reached, then the remaining codons are ignored; otherwise, it continues again from the beginning of the chromosome. The final program based on the just-described example is in this case $x \cdot y$ (see Fig. 11.4). For a fully detailed description of GE principles, see [29] or consult [http://www.grammaticalevolution.org/].

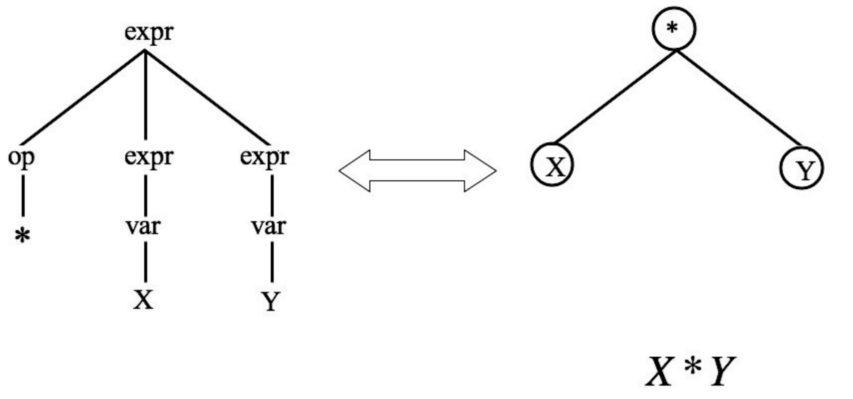


Fig. 11.4 Final program by GE

11.4.3 Analytic Programming

The final method described here and used for experiments in this chapter, is called Analytic Programming (AP), which has been compared to GP with very good results (see, for example, [52], [31], [53], [32]), [54] or visit the online university website *www.ivanzelinka.eu*, subpage *sites*.

The basic principles of AP were developed in 2001 and first published in [46],[47]. AP is also based on the set of functions, operators and terminals, which are usually constants or independent variables alike, for example:

- **functions**: sin, tan, $tanh$, And, Or,...
- **operators**: +, -, $\times$, /, dt,...
- **terminals**: 2.73, 3.14, t,...

All these objects create a set, from which AP tries to synthesize an appropriate solution. Because of the variability of the content of this set, it is called a general functional set (GFS). The structure of GFS is nested, i.e., it is created by subsets of functions according to the number of their arguments (Fig. 11.5). The content of GFS is dependent only on the user. Various functions and terminals can be mixed together. For example, GFS_{all} is a set of all functions, operators and terminals, GFS_{3arg} is a subset containing functions with maximally three arguments, GFS_{0arg} represents only terminals, etc. (Fig. 11.5).

AP, as further described later, is a mapping from a set of individuals into a set of possible programs. Individuals in population and used by AP consist of non-numerical expressions (operators, functions,...), as described above, which are in the evolutionary process represented by their integer position indexes (Fig. 11.6, Fig. 11.7, see also Chapter 2). This index then serves as a pointer into the set of expressions and AP uses it to synthesize the resulting function-program for cost function evaluation.

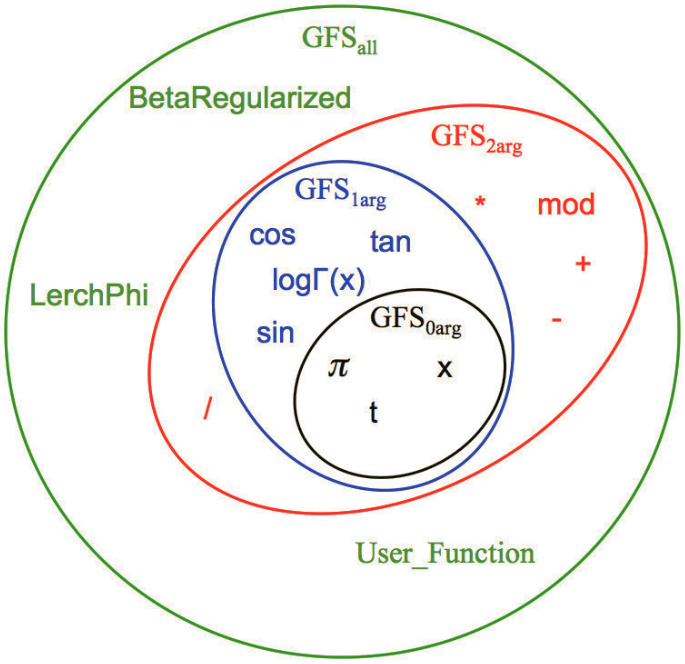


Fig. 11.5 Hierarchy in GFS

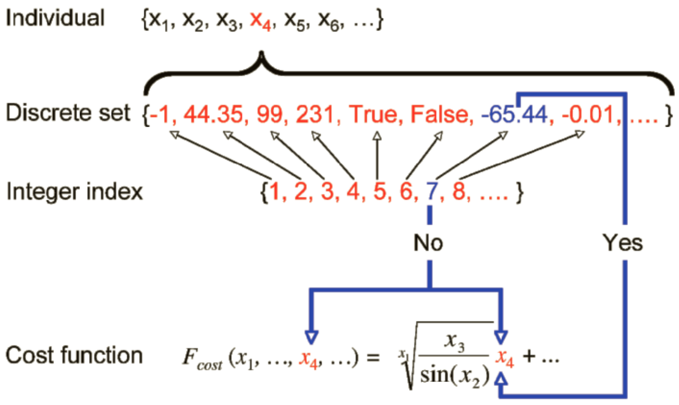

Individual
{x1, x2, x3, x4, x5, x6, …}
Discrete set
{-1, 44.35, 99, 231, True, False, -65.44, -0.01, …. }
Integer index
{1, 2, 3, 4, 5, 6, 7, 8, …. }
No
Yes
Cost function
F_cost(x1, …, x4, …) = x1√(x3/sin(x2)) x4 + …

Fig. 11.6 DSH-Integer index, see Chapter 2

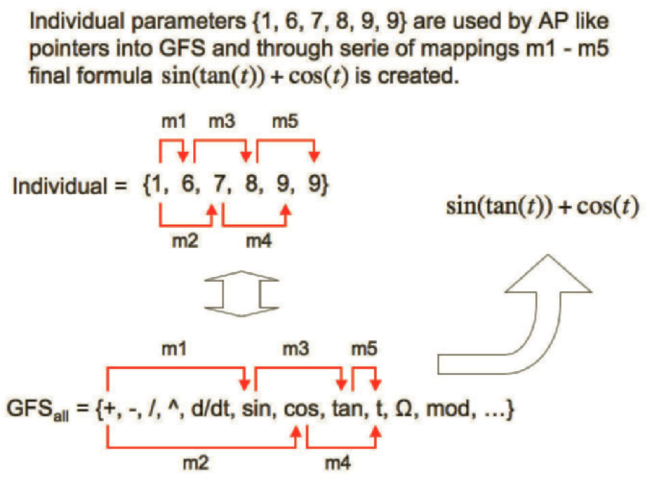

Individual parameters {1, 6, 7, 8, 9, 9} are used by AP like pointers into GFS and through serie of mappings m1 - m5 final formula sin(tan(t)) + cos(t) is created.
m1 m3 m5
Individual = {1, 6, 7, 8, 9, 9}
m2 m4
sin(tan(t)) + cos(t)
m1 m3 m5
GFSall = {+, -, /, ^, d/dt, sin, cos, tan, t, Ω, mod, ...}
m2 m4

Fig. 11.7 Principle of mapping from GFS to programs

Fig. 11.7 demonstrates an artificial example as how a final function is created from an integer individual via DSH. Number 1 in the position of the first parameter means that the operator "+" from GFS_{all} is used (the end of the individual is far enough). Because the operator "+" must have at least two arguments, the next two index pointers 6 (*sin* from GFS) and 7 (*cos* from GFS) are dedicated to this operator as its arguments. The two functions, *sin* and *cos*, are one-argument functions, so the next unused pointers 8 (*tan* from GFS) and 9 (*t* from GFS) are dedicated to the *sin* and *cos* functions. As an argument of *cos*, the variable *t* is used, so this part of the resulting function is closed (*t* is zero-argument) in its AP development. The one-argument function *tan* remains, and because there is one unused pointer 9, *tan* is mapped on *t* which is on the 9th position in GFS.

To avoid synthesis of pathological functions, a few security "tricks" are used in AP. The first one is that GFS consists of subsets containing functions with the same or a smaller number of arguments. The nested structure (see also Fig. 11.5) is used in the special security subroutine, which measures how far the end of an individual is and, according to this, mathematical elements from different subsets are selected to avoid pathological functions synthesis. More precisely, if more arguments are desired then a possible function (the end of the individual is near) will be replaced by another function with the same index pointer from the subset with a smaller number of arguments. For example, it may happen that the last argument for one function will not be a terminal (zero-argument function). If the pointer is longer than the length of subset, e.g., a pointer is 5 and is used GFS_0, then the element is selected according to the "formula" element = pointer_value mod number_of_elements_in_GFS_0. In this example, the selected element would be the variable *t* (see GFS_0 in Fig. 11.5).

GFS doesn’t need to be constructed only from clear mathematical functions as demonstrated above, but may also be constructed from other user-defined functions, e.g., logical functions, functions which represent elements of electrical circuits or robot movement commands, linguistic terms, etc.

During the evolution of a population, a different set of operators are used, such as crossover and mutation. In comparison with GP or GE, evolutionary operators like mutation, crossover, tournament, and selection are fully used in the competence of the established evolutionary algorithm. AP does not contain them in any scale of its internal structure. AP is created to be like a superstructure of EA for symbolic regression independent of their algorithmic structures. Operations used in EA are not influenced by AP and vice versa. For example, if DE is used for symbolic regression in AP, then all evolutionary operations are completed according to the DE rules and only by DE. AP just transforms individuals into formulas.

During the evolution, more or less appropriate individuals are synthesized. Some of these individuals are used to reinforce the evolution towards a better solution synthesis. The main idea of reinforcement is based on the addition of the just-synthesized and partly successful program into an initial set of terminals. Reinforcement is based on a user-defined criterion used in decision as to which individual will be used as an addition into the initial set of terminals. A criterion for the decision is in fact a threshold, i.e., by a user-defined cost value, under which conditions are synthesized solutions added into GFS. For example, if the threshold is set to 5, and if the fitness of all individuals (programs in the population) is bigger than 5, then evolution is running on the basic, i.e., initially defined, GFS. When the best individual in the actual population is less than 5, then it is entirely added into the initial GFS and is marked as a terminal. Since this moment, evolution is running on the enriched GFS containing a partially successful program. Thanks to this advantage, evolution is able to synthesize the final solutions much faster than the AP without reinforcement. This fact has been repeatedly verified by simulations on different problems. When the program is added into GFS, the threshold is also set to its fitness. If furthermore an individual with lower fitness than the just-reset threshold is synthesized, then the old one is rewritten by the better one, and the threshold is rewritten by a new fitness value again.

Adding a partially successful program as a terminal, just for simplicity, can avoid programming difficulties if it would be added like a new function. It is quite similar to automatically defined functions (ADF) for GP; however, the set of functions and terminals in GP can contain more than one ADF, which of course at least theoretically increases the complexity of the search space to the order of $n!$), including properly defined arguments of these ADF and critical situation checking (selfcalling,...). This is not a problem of AP reinforcement, because adding a program into the initial GFS is regarded as a terminal (or a terminal structure), i.e., no function, no arguments, no selfcalling, etc., and the cardinality of the initial GFS set increases only by one.

For more exact description with more details, we recommend to read [54].

11.5 Experiment Design

11.5.1 Parameter Setting

The control parameter settings of used EAs (with abbreviation in Table 11.3) have been found empirically and are given in Table 11.4 (SOMA), Table 11.5 (DE), Table 11.6 (GA), Table 11.7 (ES) and Table 11.8 (SA) respectively. The main criterion for this setting is to keep the setting of parameters as much the same as possible for all simulations and, of course, the same number of cost function evaluations as well as the same population sizes (parameter PopSize for SOMA and GA, and NP for DE, etc). Individual length represents the number of optimized parameters, i.e., in the case of this research a maximal number of integer indexes will be used in evolution in order to synthesize a new chaotic system.

AP, being applied on the evolutionary algorithms in 13 versions, was used for the experimentation. Symbolic objects (e.g., variables, constants,...) for manipulation and complex structure synthesis were selected from the well-known logistic equation:

$$x_{n+1} = Ax(1-x) \tag{11.6}$$

Table 11.3 Algorithms abbreviation

Algorithm	Version	Abbreviation
SOMA	AllToOne	A
	AllToOneRandomly	B
	AllToAll	C
	AllToAllAdaptive	D
Differential Evolution	DERand1Bin	E
	DERand2Bin	F
	DEBest2Bin	G
	DELocalToBest	H
	DEBest1JIter	I
	DERand1DIter	J
Genetic Algorithm		K
Evolutionary strategies (μ,λ)		L
Simulated annealing		M

Table 11.4 SOMA setting for 4 basic search strategies: A, B, C, D

Algorithm	A	B	C	D
PathLength	3	3	3	3
Step	0.11	0.11	0.11	0.11
PRT	0.1	0.1	0.1	0.1
PopSize	200	200	200	200
Migrations	10	10	10	10
MinDiv	-0.1	-0.1	-0.1	-0.1
Individual Length	50	50	50	50

Table 11.5 DE setting for 6 basic search strategies: E, F, G, H, I, J

Algorithm	E	F	G	H	I	J
NP	200	200	200	200	200	200
F	0.9	0.9	0.9	0.9	0.9	0.9
CR	0.3	0.3	0.3	0.3	0.3	0.3
Generations	200	200	200	200	200	200
Individual Length	50	50	50	50	50	50

Table 11.6 GA setting for canonical version of GA: K

Algorithm	K
PopSize	200
Mutation	0.4
Generations	100
Individual Length	50

Table 11.7 ES setting for search strategy: L

Algorithm	L
μ ,λ	200
σ	0.8
Iterations	100
Individual Length	50

Table 11.8 SA setting for search strategy: M

Algorithm	M
No. of particles	200
σ	0.5
$k_{\max}$	66
T_{min}	0.0001
T_{max}	1000
α	0.95
Individual Length	50

This selection was based on the fact that the logistic equation is a well-known simple system that can produce chaotic behavior. This equation is also well analyzed. It was expected that evolutionary search would be able to synthesize the logistic equation, which was in fact a source of elements for GFS. Evolutionary synthesis of logistic equation was actually observed, as further discussed later. Another reason behind the selection of the logistic equation is that results from designed experiments can be easily compared, verified and analyzed.

The basic set of objects used in symbolic regression are $\{x, A, +, -, \times, /\}$. It is also important to note that experiments provided here, i.e., evolutionary synthesis of chaotic systems, are not restricted to one-dimensional chaotic maps but can be applied in principle to synthesis of higher-dimensional and more complex chaotic systems. This declaration is based on many other successful complex examples accomplished by GP, GE and AP in the past.

In this research, a total of 1300 independent simulations were completed, 100 trials by each of the 13 algorithms. Each simulation was started at randomly selected initial conditions (i.e., each initial population was randomly generated).

11.5.2 Cost Function

The cost function was in fact a little bit complex decision function with multiple "If" conditions.

The cost function used for chaos synthesis, comparing with other problems like chaos control [49], [50] or black-box optimization [28], is quite a complex structure which cannot be easily described by a few simple mathematical equations. Instead, it is described by the following procedure:

1. Take a synthesized function and evaluate it for 500 iterations with a sampling step of Δ A = 0.1.
2. Check if each value of A for all 500 iterations is unique or if some data are repeated in the series (the first check for chaos, indirectly). If the data are not unique, then go to step 5, else go to step 3.
3. Take the last 200 values, and for each value of A, calculate its Lyapunov exponent.
4. Check the Lyapunov exponent: If the Lyapunov exponent is positive, write all important data (synthesized functions, number of cost function evaluations, etc.) into a file. Then, repeat the simulation for another synthesized system by going to step 1.
5. If the data are not unique, i.e., if the Lyapunov exponent is not positive, return an individual fitness, and sum all values whose occurrences in the dataset from step 1 are more than 1 (simply, it returns the occurrences of periodicity, quasi-periodicity – higher penalization of an individual in the evolution).

More brief and simple description of above algorithmically defined cost function can be also done as in eq.11.7.

$$\begin{aligned}&Data[f_{synt,\,1},\ldots,f_{synt,\,500}] := f_{synt,\,k+1} = f_{synt,\,k}(x_{start}),\ k \in [1,\ 500]\\&\begin{cases} if\ Data[f_{synt,\,1}] \neq Data[f_{synt,\,2}] \neq \ldots. \neq Data[f_{synt,\,500}] \\ then\ \{\text{calculate } \lambda \text{ for } Data[f_{synt,\,300},\cdots,f_{synt,\,500}]\ ,\ if\ \lambda > 0 \text{ write all to file} \\ else \text{ penalize individual} \end{cases}\end{aligned} \tag{11.7}$$

The input to this cost function is a synthesized function and the output is the fitness (quality) of the synthesized function (i.e., the individual in the population).

In the cost function, it was tested twice to ensure that the behavior of the just-synthesized formula is really chaotic. The first test was done in step 2 (unique appearance in the data series) and the second one, in step 4, where the Lyapunov exponent was tested numerically [16].

The reason as to why in step 5, the sum of the non-unique data appearances was returned is based on the fact that the evolution is searching for minimal values. In this case, the value 2 means that some data element appears in the 500-data series twice, and 1 would means that there is no periodicity and thus synthesized system is a possible candidate for chaos.

To ensure that the results obtained are correct, all written synthesized functions were used for automatic generation of bifurcation diagrams and Lyapunov exponents, as further discussed below.

11.5.3 Case Studies

Two case studies are presented in this chapter. The first and main one (discrete systems) is the continuation of research done in [54]. Simulations has been enlarged (compare with [54]) for other evolutionary algorithms (GA, SA, ES). The second one is focused on how to synthesize simple discrete systems based on user demand.

11.5.3.1 Discrete Systems: Simulations and Results

All algorithms (SOMA, DE, GA, ES and SA) in 13 versions have been applied for 100 times in order to find artificially synthesized functions that can produce chaos. All of these experiments were done using the *Mathematica* software. The primary aim of this comparative study is not to show which algorithm is better or worst, but to show that symbolic regression is able to synthesize some new (at least in the sense of mathematical description and behavior) chaotic systems.

Based on the results from experiments, two different sets of figures were created. The first set (Fig. 11.8 - Fig. 11.10) shows the performances of different algorithms from different points of views, the second set (Fig. 11.13 - Fig. 11.88) shows behaviors of the selected synthesized programs, i.e., bifurcation diagrams. The synthesized programs are also appended to each figure in the form of the mathematical formula. Fig. 11.11 shows an example of the so-called program length histogram, generated from 100 simulations. Program length (in *Mathematica* command: LeafCount, denoted as LC) means a number of elements that create a mathematical formula.

As an example, the logistic equation (11.6), for which LC = 8, seems at the first glance to be false (equation contains $A, x, 1$ and $\times$). However, with a closer look at this equation via the *Mathematica* command *TreeForm*, one gets formula 11.8 (see also Fig. 11.12), and LC = 8 is thus clear: ($\times$, A, +, 1, $\times$, -1, *x, x*).

$$Times[A, Plus[1, Times[-1, x]], x] \tag{11.8}$$

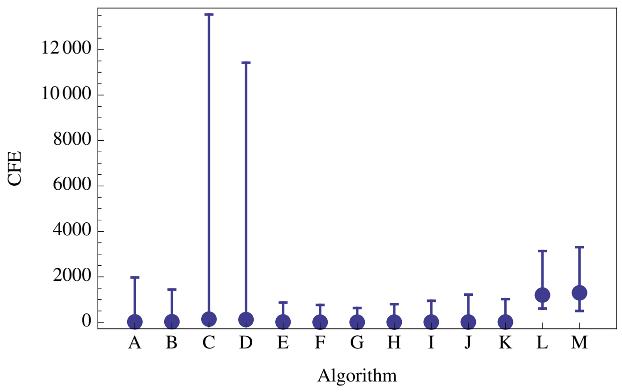


Fig. 11.8 Mutual comparison of all algorithms

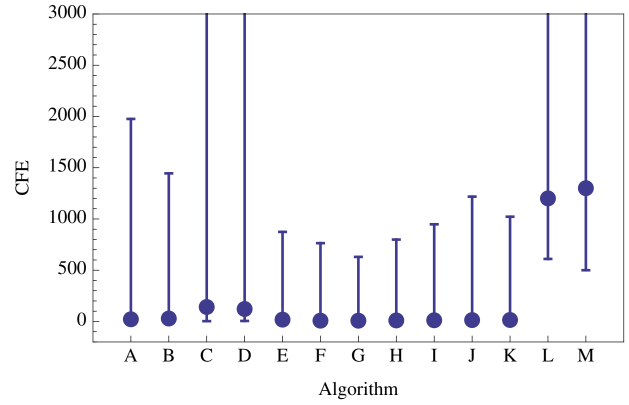


Fig. 11.9 Mutual comparison of all algorithms - detail

There is an explanation for the contradiction between the fact that the length of an individual was set to 50 (Table 11.4 - Table 11.8) and Fig. 11.10, where one can observe programs of lengths larger than 50. The explanation is that when a symbolic string like "$Ax(1-x)$" is transformed into an expression, it becomes a formula 11.8, i.e., it is "artificially" enhanced due to some *Mathematica* internal programming reasons.

For a better overview of the performances of all such algorithms and the lengths of the synthesized programs, Fig. 11.9 was generated and displayed, where for all 13 algorithms, the corresponding minimal, average and maximal values are depicted. Almost the same quality in LC is observed for all of them.

When evolutionary techniques are used, usually their performances are evaluated via cost function evaluations [2], [48], i.e., how many times the cost function has to be re-calculated in order to reach a suitable solution. Fig. 11.8 and Fig. 11.9 are

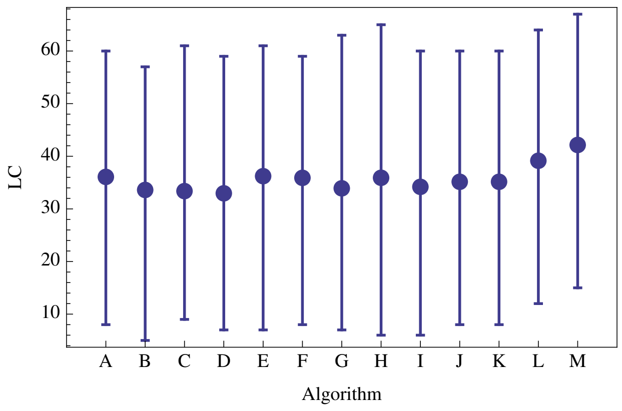


Fig. 11.10 Mutual comparison of LC of all algorithms

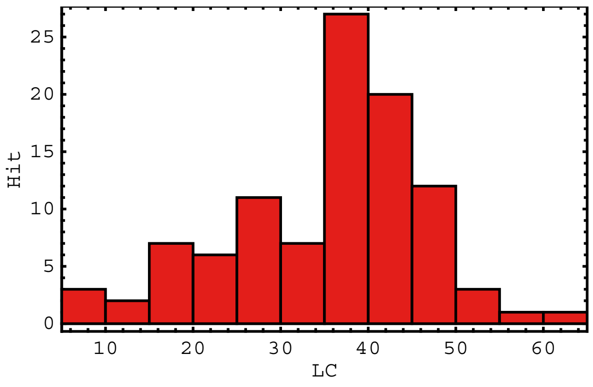


Fig. 11.11 Histogram of LC for DERand1Bin

displayed for this purpose. From both pictures, it appears that only algorithms C and D have less performance. But this is not true, because the SOMA versions C and D have larger numbers of cost function evaluations (see [48]). Because EA was stopped only according to a defined number of cost function evaluations (see [48] or Section 11.5.1 Parameter Setting), these two versions of SOMA logically differ from each other, as shown in Fig. 11.9 .

For mutual comparisons of algorithm performances in successfully generating chaotic systems, Fig. 11.13 - Fig. 11.88 and corresponding equations show selected (visually the best) results of all 1300 simulations in all case studies. With each figure is joined equations of the synthesized systems. To be "sure" that those bifurcation diagrams are true, Lyapunov exponents were also generated for each bifurcation diagram (not completely reported here).

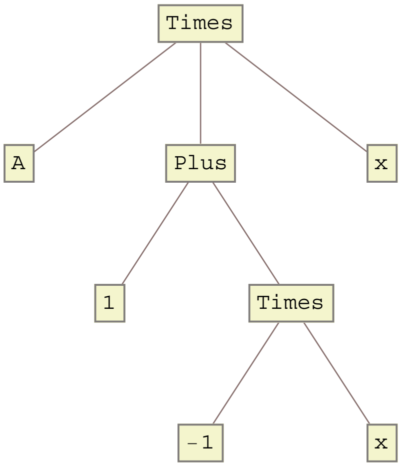
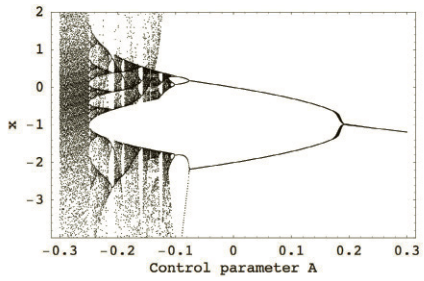


Fig. 11.12 Tree representation of eq. (11.8)



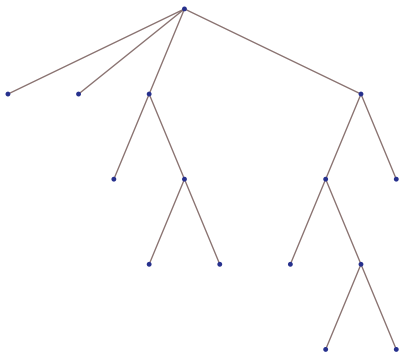
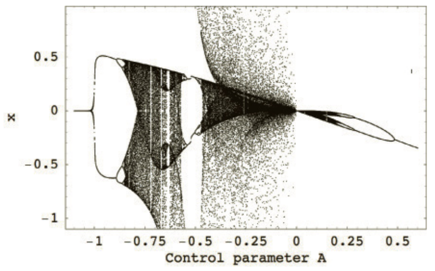


Fig. 11.13 Bifurcation diagram of $\frac{2A(2x-1)}{A+x^2}$ ...

Fig. 11.14 ... and its tree representation.



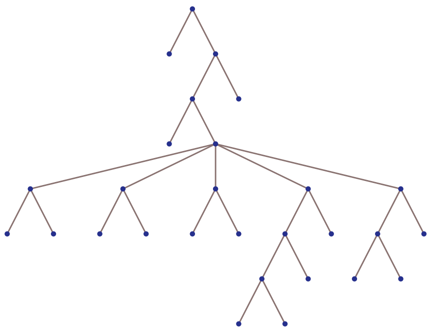


Fig. 11.15 Bifurcation diagram of $\frac{x}{\frac{x^3}{2A^3(x-A)(A+x)}+A}$ ...

Fig. 11.16 ... and its tree representation.

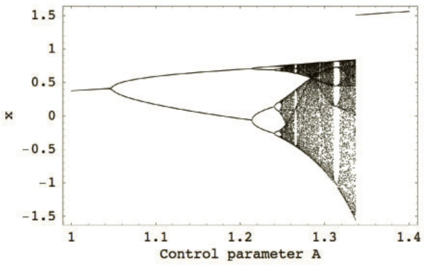


Fig. 11.17 Bifurcation diagram of $A - \frac{A}{x\left(A(-x)+\frac{A}{x}-\frac{x(Ax+A+x)+1}{A}+A\right)+A}$ ...

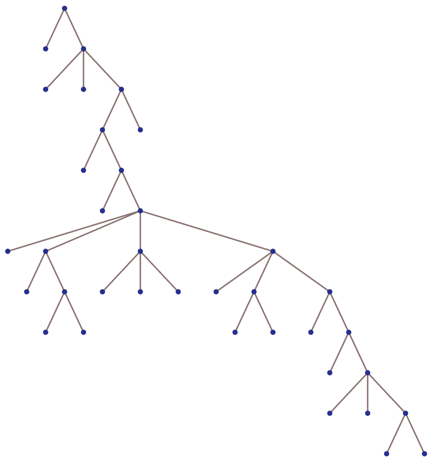


Fig. 11.18 ... and its tree representation.

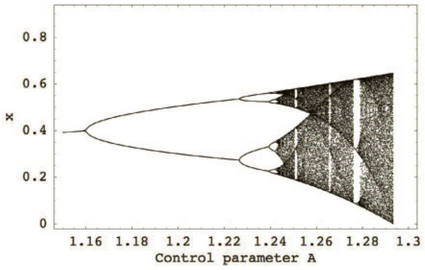


Fig. 11.19 Bifurcation diagram of $\frac{A\left(\frac{A+x}{2A+1}+2A\right)}{\frac{1}{x(A-2x)}+A}$ ...

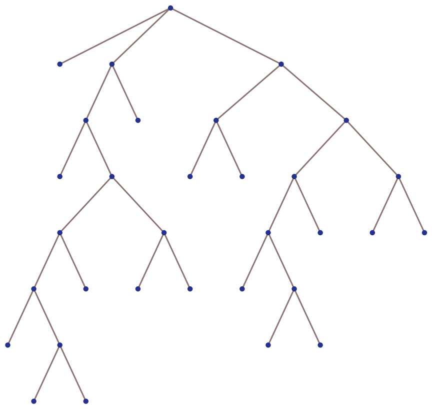


Fig. 11.20 ... and its tree representation.

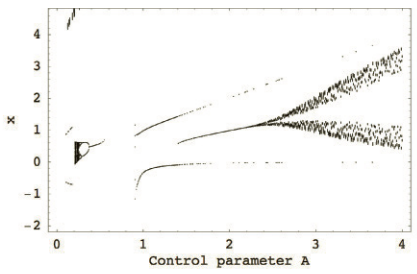


Fig. 11.21 Bifurcation diagram of $-\frac{x}{(A+x)\left(A^2+\frac{x}{A}\right)(2(A-1)x+x^2)-A}+A-x$ ...

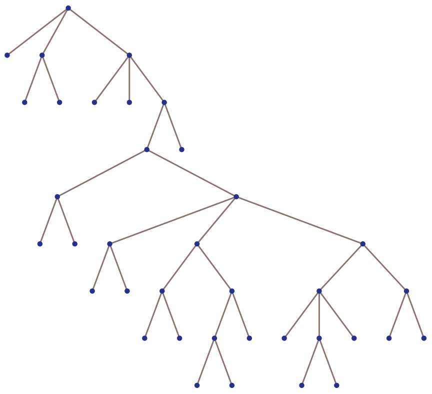


Fig. 11.22 ... and its tree representation.

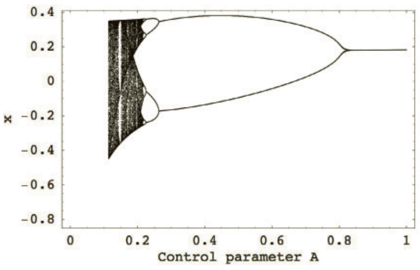


Fig. 11.23 Bifurcation diagram of $\frac{x\left(3Ax^2-Ax+\frac{A}{x}-2A-2x\right)}{3A+x^2}$ ...

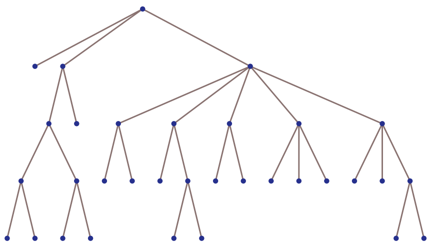


Fig. 11.24 ... and its tree representation.

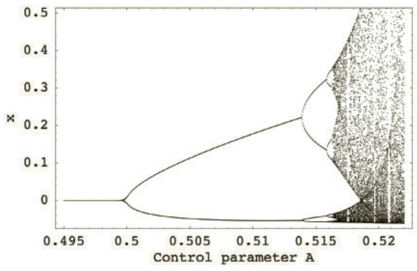


Fig. 11.25 Bifurcation diagram of $\frac{x}{A(-x)+\frac{x(-A-x+1)+A}{A+x}-\frac{1}{A(A-x)}+4A-x}$ ...

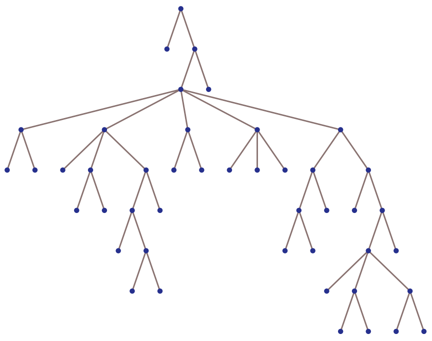


Fig. 11.26 ... and its tree representation.

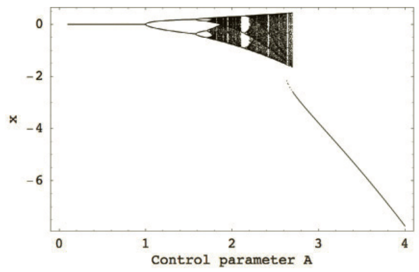


Fig. 11.27 Bifurcation diagram of $\frac{A(A(-x)-A-x)}{\frac{A}{x}+\frac{x}{A}+2x}$ ...

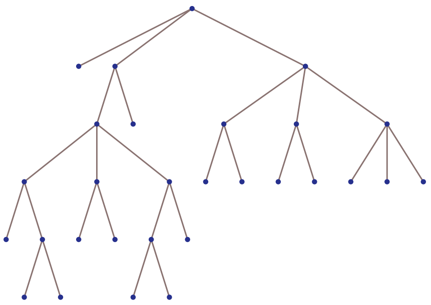


Fig. 11.28 ... and its tree representation.

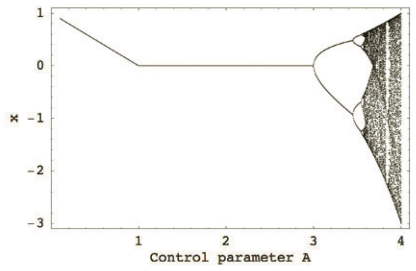


Fig. 11.29 Bifurcation diagram of $2x - x(A+x)$ ...

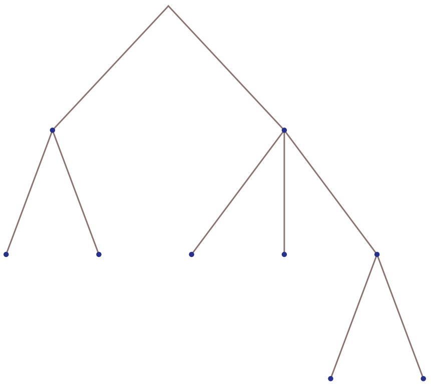


Fig. 11.30 ... and its tree representation.

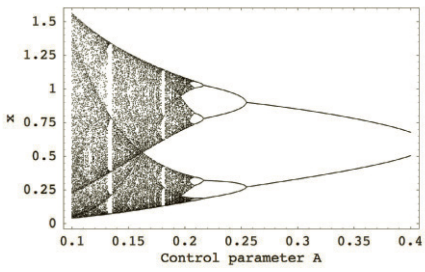


Fig. 11.31 Bifurcation diagram of $\frac{A}{\frac{x\left(\frac{A^2}{x^2}+Ax^2\right)}{-x(A+x)+\frac{x}{A}+2A-x}+x^2}$ ...

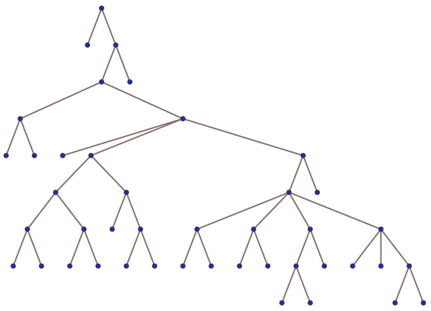


Fig. 11.32 ... and its tree representation.

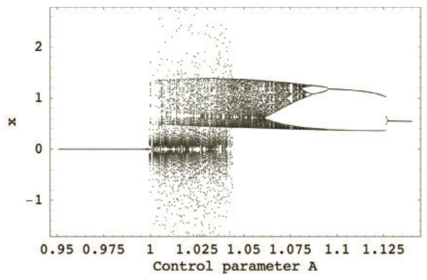


Fig. 11.33 Bifurcation diagram of $\frac{A}{\frac{-\frac{-A^2+\frac{2A+x}{A}-x}{x}-\frac{x}{3A-2x}+A}{A}+Ax+A+x}$ ...

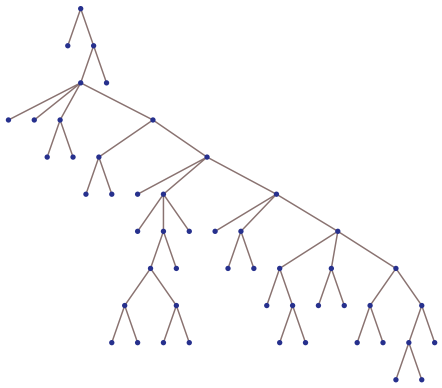


Fig. 11.34 ... and its tree representation.

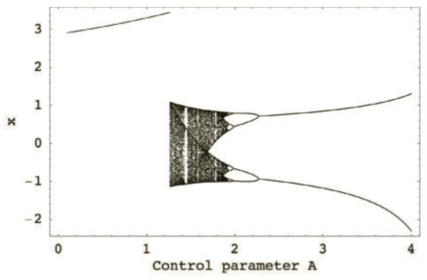


Fig. 11.35 Bifurcation diagram of $\frac{-A+3x-\frac{1}{x}}{\frac{A}{x}-A+x}$ ...

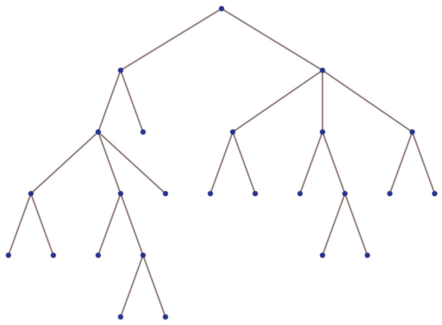


Fig. 11.36 ... and its tree representation.

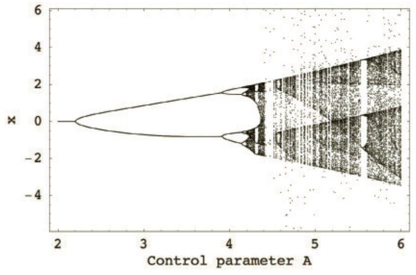


Fig. 11.37 Bifurcation diagram of $\frac{A^2}{-\frac{x^2}{A}+\frac{1}{2x\left(x^2-A\right)}-\frac{A}{x}-2x}+x$ ...

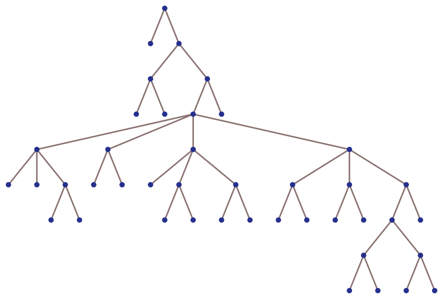


Fig. 11.38 ... and its tree representation.

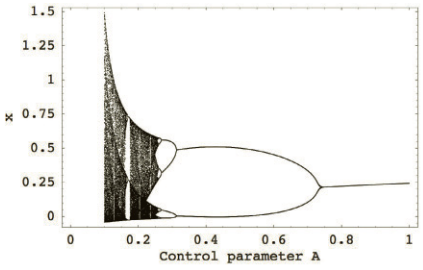


Fig. 11.39 Bifurcation diagram of $\frac{(A-x)\left(-\frac{Ax^2}{A+x}+A-x^2\right)}{(A+x)(2A+x)}$ ...

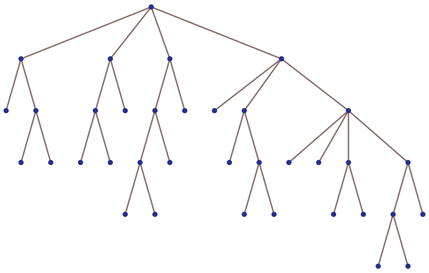


Fig. 11.40 ... and its tree representation.

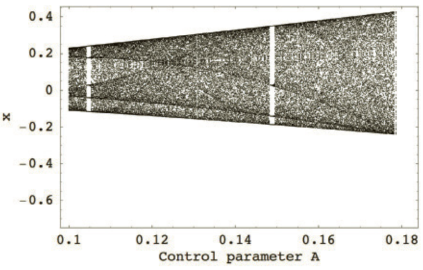


Fig. 11.41 Bifurcation diagram of $\frac{(-Ax-x)(Ax-2A+x)}{\frac{Ax}{A+x}+x^2}$ ...

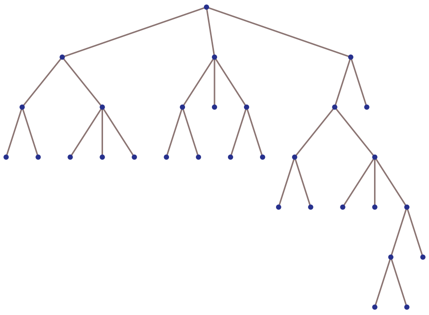


Fig. 11.42 ... and its tree representation.

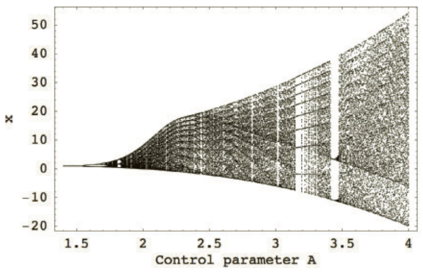


Fig. 11.43 Bifurcation diagram of $\frac{A^2(A-Ax)}{\frac{x\left(-\frac{A}{x}+x+1\right)}{A}+A}+x$ ...

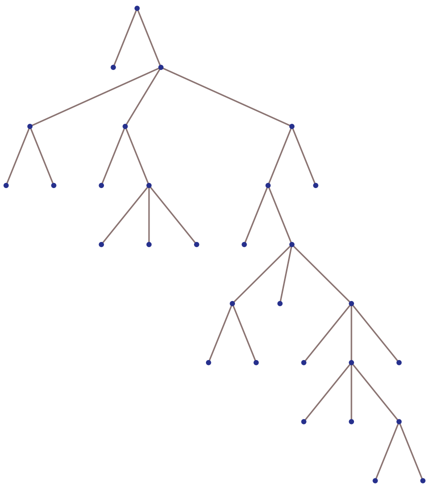


Fig. 11.44 ... and its tree representation.

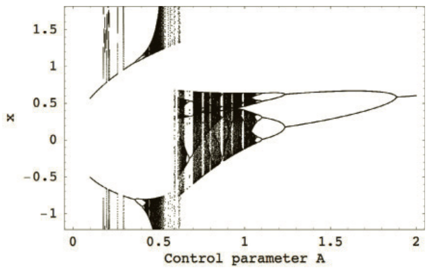


Fig. 11.45 Bifurcation diagram of $\frac{A}{\frac{x^2}{A}-x\left(\frac{x}{x-A}+A-x\right)-\frac{3x-A}{A+2x}+A}-x$ ...

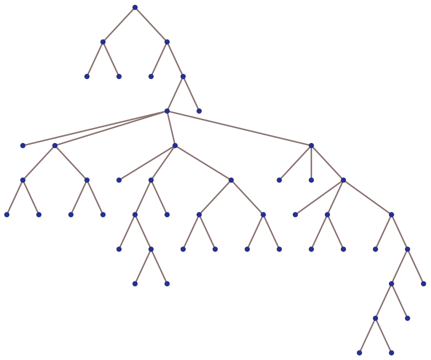


Fig. 11.46 ... and its tree representation.

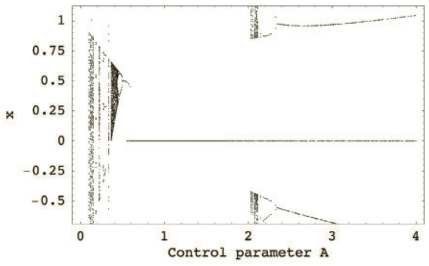


Fig. 11.47 Bifurcation diagram of $\frac{x}{\frac{x\left((A-x)^2-2x\right)}{-\frac{2A}{\frac{x}{A}-1}-A}+A}-x$ ...

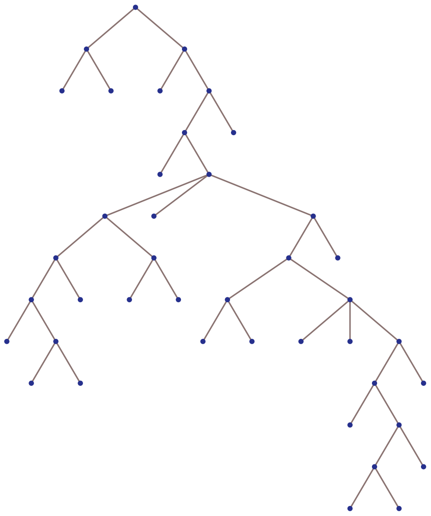


Fig. 11.48 ... and its tree representation.

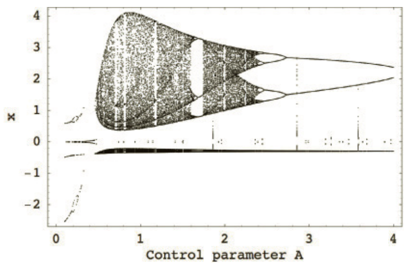


Fig. 11.49 Bifurcation diagram of $\frac{A+x}{-\frac{A}{x}-x\left(\frac{3A+x}{x}+x\right)}$ ...

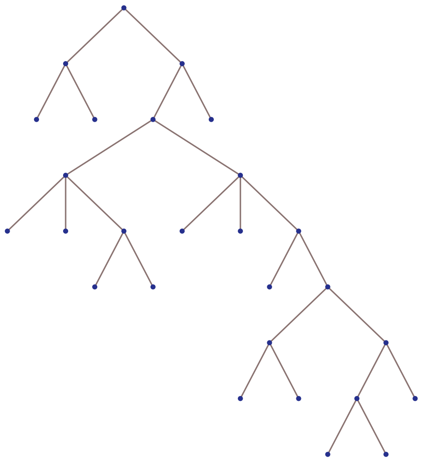


Fig. 11.50 ... and its tree representation.

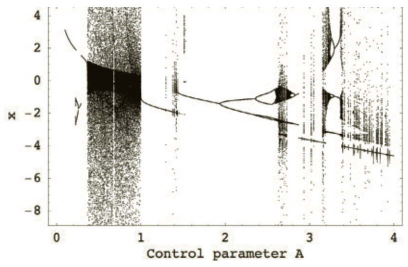


Fig. 11.51 Bifurcation diagram of $\frac{\left(Ax+\frac{x}{A}+A+x^2\right)\left(-\frac{2A}{A+x}+A+x\right)}{x(Ax-x(A-x)+2A+x)}-A$ ...

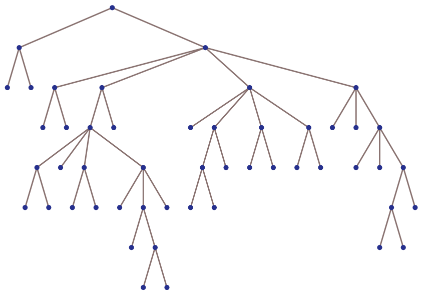


Fig. 11.52 ... and its tree representation.

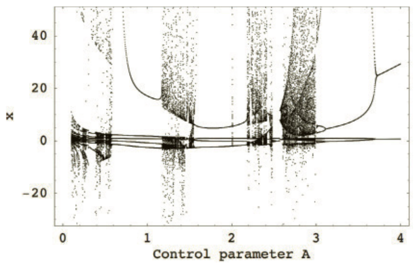


Fig. 11.53 Bifurcation diagram of $\frac{\frac{A^2(A-x)}{x(A(-x)+A+2x)}-2A+x-1}{x}$ ...

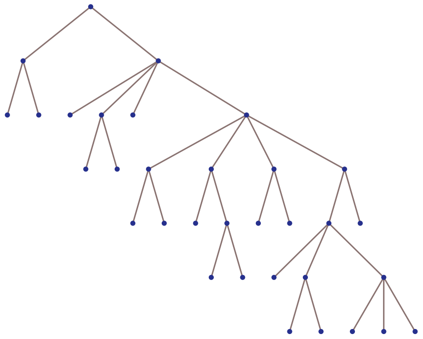


Fig. 11.54 ... and its tree representation.

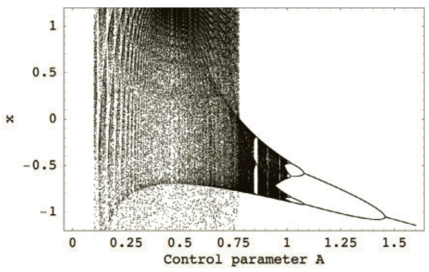


Fig. 11.55 Bifurcation diagram of $\frac{A+\frac{1}{2A}-3x}{\frac{A}{2x}-\frac{A}{A+x}-A-x}$ ...

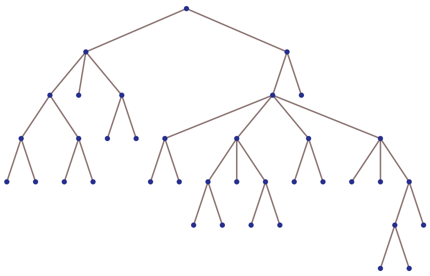


Fig. 11.56 ... and its tree representation.

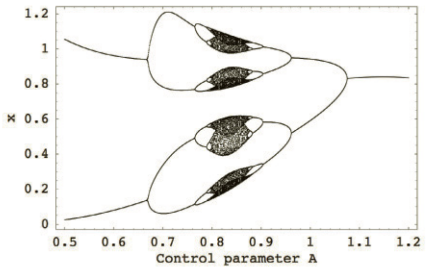


Fig. 11.57 Bifurcation diagram of $-\frac{A}{2x\left(-\frac{x^3}{A^3}-\frac{A^2}{x}+A\right)}$ ...

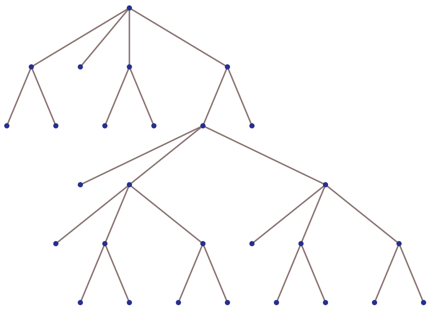


Fig. 11.58 ... and its tree representation.

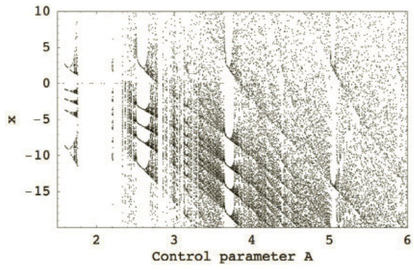


Fig. 11.59 Bifurcation diagram of $\frac{A^2x\left(A^2+A+2x-1\right)}{(-A-x+1)\left(\frac{A}{x}-A+x+\frac{1}{x}\right)}$ ...

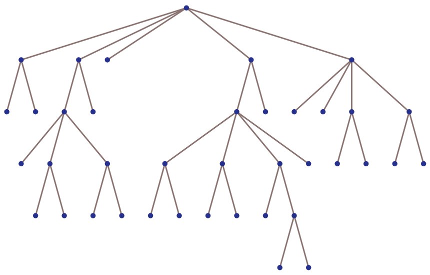


Fig. 11.60 ... and its tree representation.

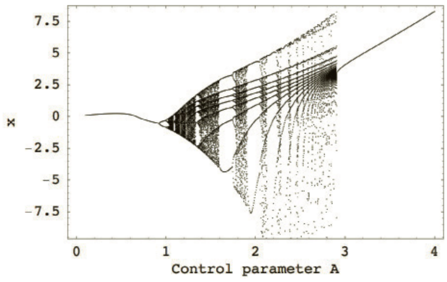


Fig. 11.61 Bifurcation diagram of $A - \frac{A\left(3A-x^2\right)}{\frac{A-2x(-A-x)}{A}+1}$ ...

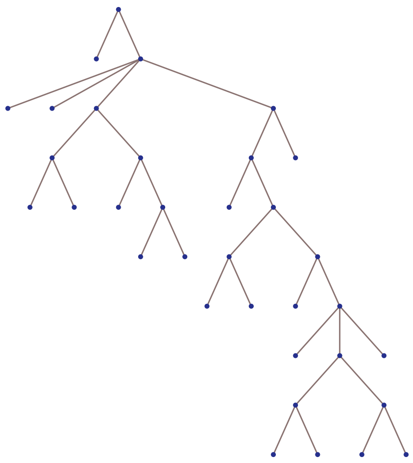


Fig. 11.62 ... and its tree representation.

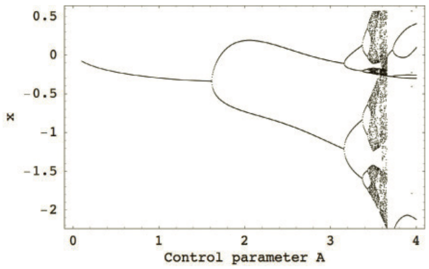


Fig. 11.63 Bifurcation diagram of $\frac{A}{\frac{\frac{A(A+x)}{2x}+\frac{A}{Ax+1}}{\frac{1}{x}+1}-2A+x-1}$ ...

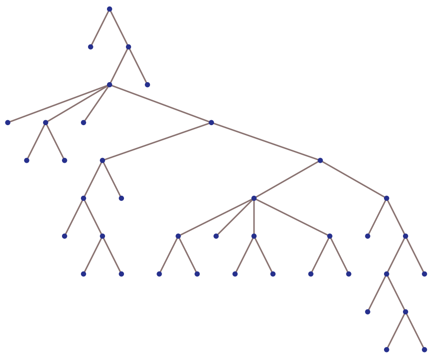


Fig. 11.64 ... and its tree representation.

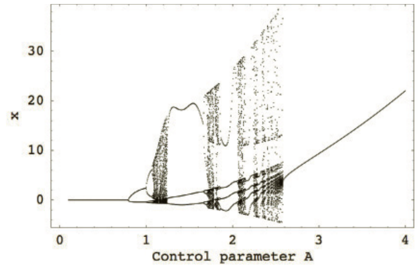


Fig. 11.65 Bifurcation diagram of $-\frac{2A^2x(A(-x)+A+x)}{Ax^2+2x+1}$ ...

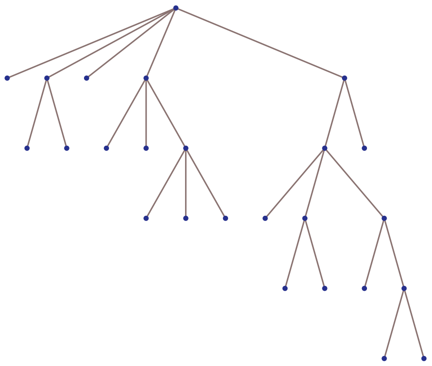


Fig. 11.66 ... and its tree representation.

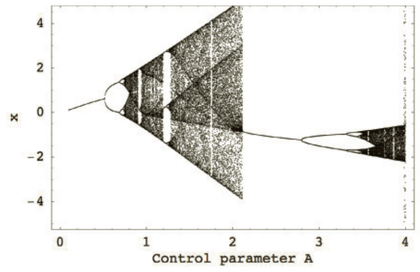


Fig. 11.67 Bifurcation diagram of $\frac{\frac{A}{x}+2A-x^2}{\frac{A+x}{2Ax}+x}$ ...

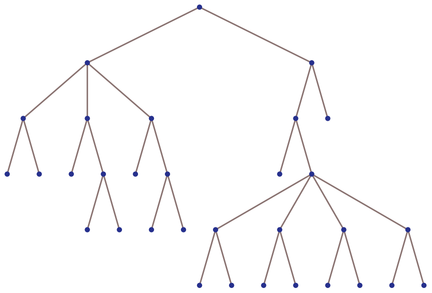


Fig. 11.68 ... and its tree representation.

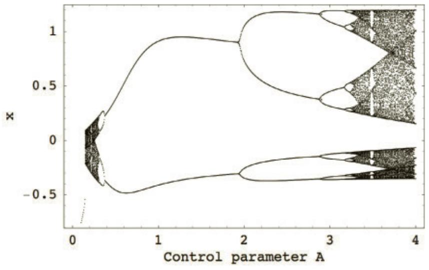


Fig. 11.69 Bifurcation diagram of $\frac{Ax-A-x}{\frac{A}{2x}-\frac{A(A-x)}{Ax+2A-x+1}+A+x}$ ...

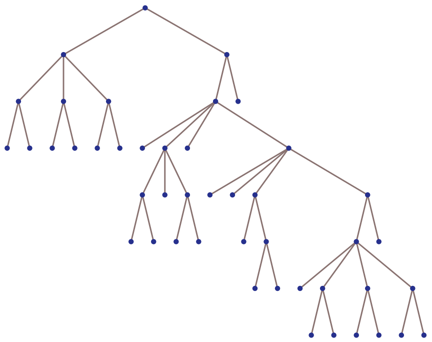


Fig. 11.70 ... and its tree representation.

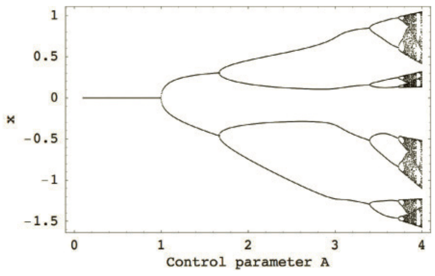


Fig. 11.71 Bifurcation diagram of $\frac{A}{\frac{x^2\left(Ax+\frac{x}{A}\right)}{x-A}-\frac{4Ax}{A+2x}+2x-\frac{1}{x}}$ ...

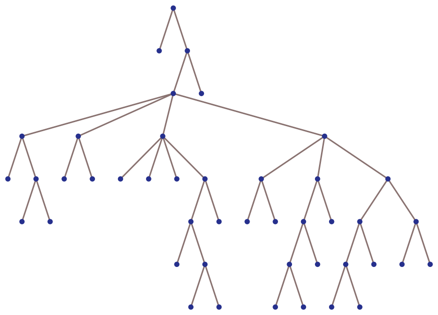


Fig. 11.72 ... and its tree representation.

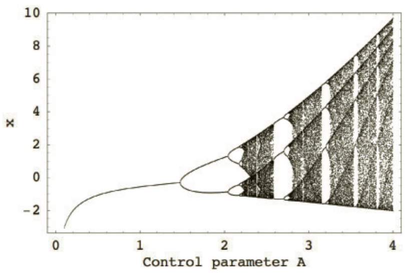


Fig. 11.73 Bifurcation diagram of $-\frac{A(A-x)(2x-A)}{2A+\frac{1}{A}+x^2}-A+x$ ...

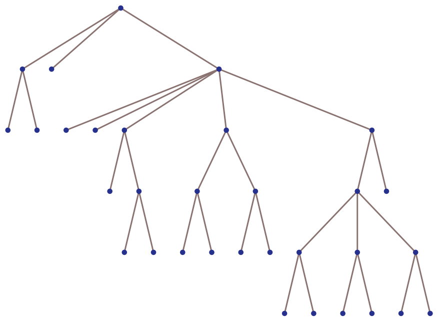


Fig. 11.74 ... and its tree representation.

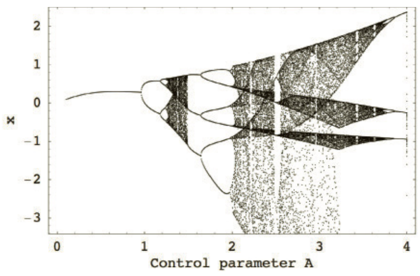


Fig. 11.75 Bifurcation diagram of $\frac{x-Ax^2(Ax+A+x)}{x^2\left(A+\frac{1}{x^2}-x\right)\left(\frac{x}{A}+x\right)}$ ...

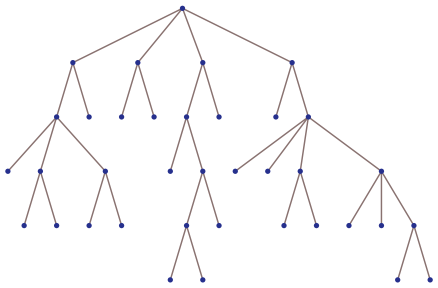


Fig. 11.76 ... and its tree representation.

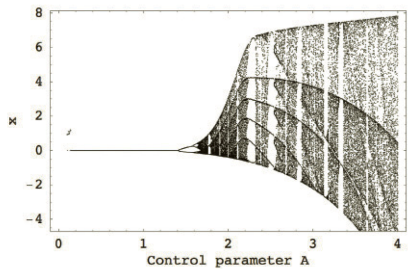


Fig. 11.77 Bifurcation diagram of $\frac{A^3}{A^2-\frac{A}{x}-\frac{(-A-x)(x-3A)}{A}-A-x}+x$ ...

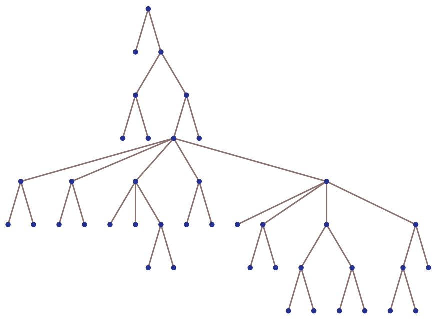


Fig. 11.78 ... and its tree representation.

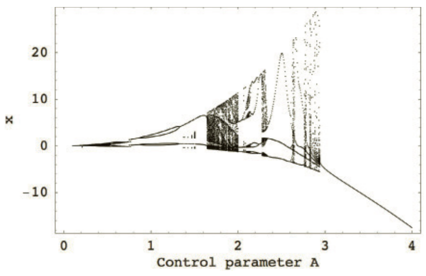


Fig. 11.79 Bifurcation diagram of $\frac{(x-A)\left(A(-x)-\frac{A}{x}-A+2x\right)}{x\left(\frac{A}{x^2\left(Ax+\frac{2x}{A}+2A\right)}+\frac{x}{2A}\right)}$ ...

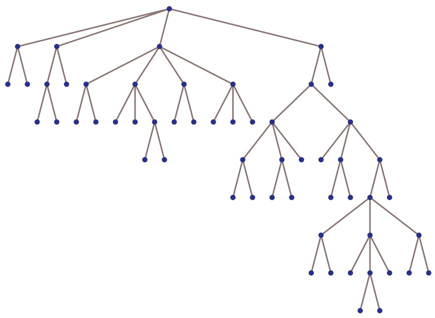


Fig. 11.80 ... and its tree representation.

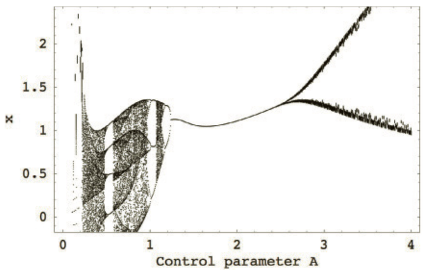


Fig. 11.81 Bifurcation diagram of $\frac{A}{A^3x(A-x)+\frac{x^3}{A^2}+\frac{A(A-x)}{2x^3}}+A-x$ ...

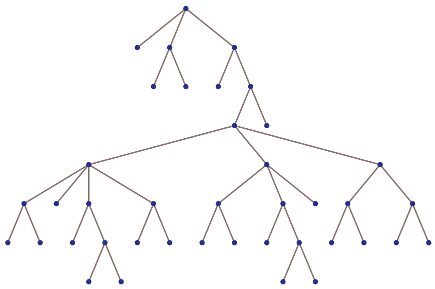


Fig. 11.82 ... and its tree representation.

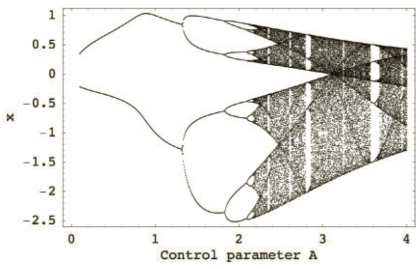


Fig. 11.83 Bifurcation diagram of $\frac{Ax-2x+2}{x^2-\frac{\frac{A}{x}+x}{A}}$ ...

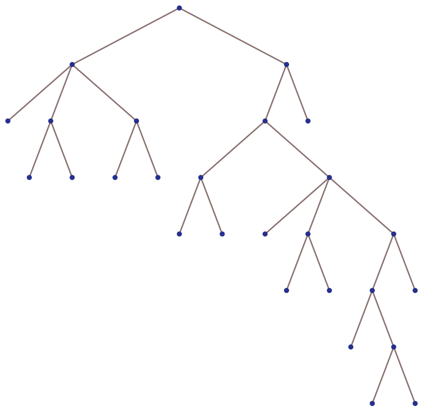


Fig. 11.84 ... and its tree representation.

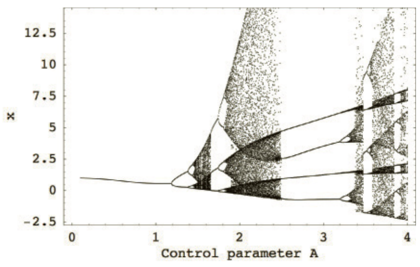


Fig. 11.85 Bifurcation diagram of $\frac{A-2A\left((A-x)^2-x\right)}{x(A-x)-\frac{2(Ax+A+x)}{A}}+1$ ...

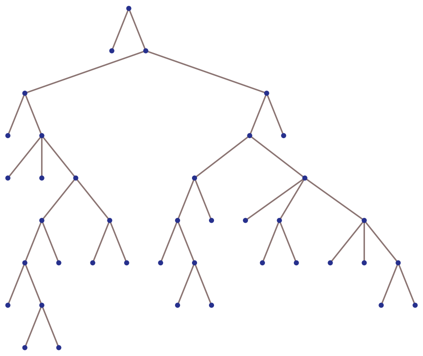


Fig. 11.86 ... and its tree representation.

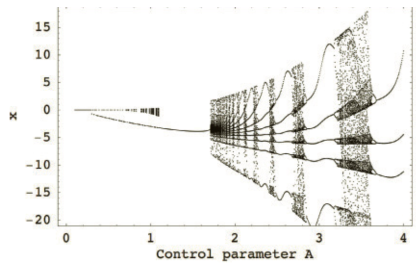


Fig. 11.87 Bifurcation diagram of $\frac{2Ax\left(-A^2-2x\right)}{A+x^2}$ ...

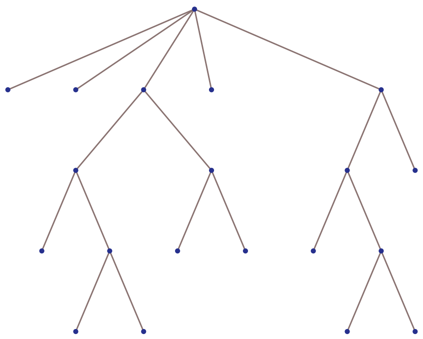


Fig. 11.88 ... and its tree representation.

11.5.3.2 Engineering Design: Preliminary Study

The same principle has been used to synthesize discrete chaotic systems with user defined conditions, i.e. simple engineering design of chaotic systems has been done in this part. In this case study, systems with chaotic behavior in the interval [2, 3], has been accepted while search has been running in interval [0, 4] (overlapping during search was also allowed). In this preliminary study only DE and SOMA algorithms have been used in 50 repeated simulations. Selected demonstrative results are depicted on Fig. 11.89 - 11.96. Basic set of objects used in symbolic regression was again $\{x, A, +, -, \times, /\}$. Bifurcation diagrams of selected systems (Fig. 11.89 - 11.96) are in a few cases overlapping interval [2, 3] and are different in amplitude. It is reasonable to expect that if restriction would be applied on amplitude, then such defined systems would also be synthesized.

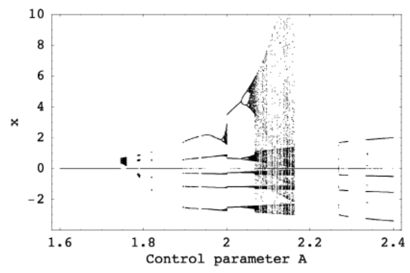


Fig. 11.89 Engineering design: bifurcation diagram of
$-\frac{A(A+2x)}{2\left(\frac{A\left(Ax+\frac{A}{x}\right)}{\left(A^2-A+1\right)(Ax+2x)(A(x-A)+A)}+x\right)}$.

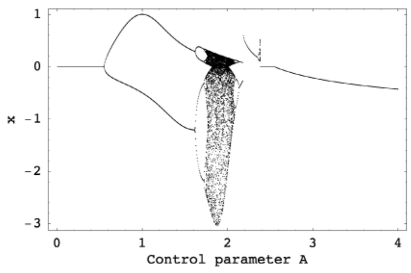


Fig. 11.90 Engineering design: bifurcation diagram of $\frac{A\left(-A^2+A+x\right)+A^2+A-x}{A^2(-x)+A(A^2+x)-\frac{A+x}{Ax}-A-2x}$.

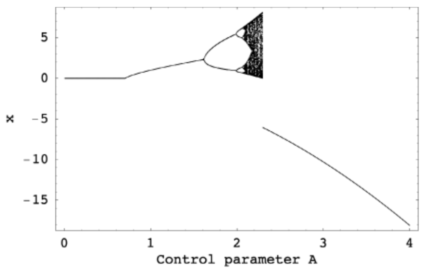


Fig. 11.91 Engineering design: bifurcation diagram of $-\frac{Ax\left(Ax-A^2(x-2A)\right)}{A(-A-x^2+x)-x}$.

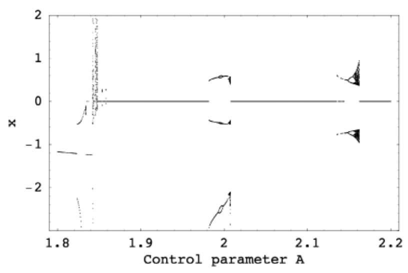


Fig. 11.92 Engineering design: bifurcation diagram of
$\frac{Ax}{\left((A+x)\left(-A^2+A+x\right)+A+1\right)\left(A-\frac{A\left(\frac{x}{2\left(-\frac{A}{x}+A-x\right)(A+x)}+\frac{x+1}{A}\right)}{2x}\right)}$.

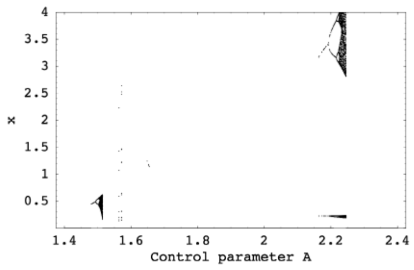


Fig. 11.93 Engineering design: bifurcation diagram of $\frac{A\left(2A(x-A)+\frac{1}{A-x}+A\right)}{-\frac{1}{A^2x^2}+\frac{4x^3}{A}-x(x-A)-x}$.

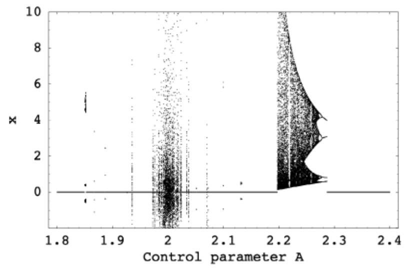


Fig. 11.94 Engineering design: bifurcation diagram of $-\frac{Ax^2}{-A^2+2A-x^3+x}$.

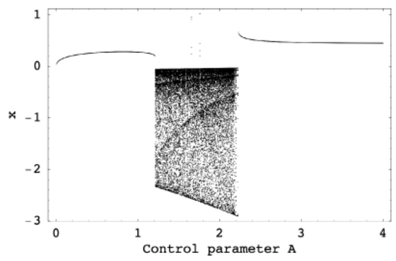


Fig. 11.95 Engineering design: bifurcation diagram of

$$\frac{A}{x\left(\frac{A}{-A^2+Ax+A-x^2+x}+Ax^2-x\right)+\frac{A}{2x}+A+x}.$$

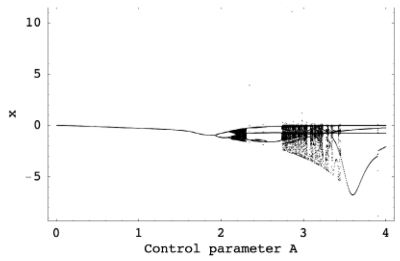


Fig. 11.96 Engineering design: bifurcation diagram of

$$\frac{A^2}{\left(Ax(A-x)+\frac{x}{A}+A-3x\right)\left(\left(\frac{x}{A}+A-x^2\right)\left(Ax^2+x\right)+\frac{Ax\left(\frac{x}{2A+2x}-Ax\right)}{\frac{A}{x(A+x)}-\frac{x}{A-x}}-A\right)-\frac{A}{A+x}-A}.$$

11.6 Conclusion

The aim of this paper is to show how various chaotic systems can be synthesized by means of evolutionary algorithms. Evolutionary synthesis of chaotic systems has been applied to 13 basic comparative simulations in this chapter. Each comparative simulation was repeated 100 times and all 1300 results (100 simulations for each algorithm) were used to create Fig. 11.14 - Fig. 11.88 for overall performance evaluation of evolutionary chaos synthesis. The results look quite promising and convincing.

For comparative studies, five algorithms was used - Differential Evolution (DE) [35], Self Organizing Migrating Algorithm (SOMA) [48], Genetic Algorithms [17], [4], Simulated Annealing (SA) and [9] Evolutionary Strategies (ES) [3]. They were chosen to show that evolutionary synthesis of chaos by AP can be implemented via any evolutionary algorithm and that they all give reasonable results.

The method of symbolic regression described in this paper is relatively simple, but feasible to implement and easy to use. Based on its principles and its possible universality (as just mentioned, it was tested with 5 evolutionary algorithms – SOMA, DE, GA, ES and SA in 13 versions), symbolic regression seems quite capable of synthesizing new dynamical systems for generating chaos.

As a summary, the following statements are presented:

- **Result verification.** To be sure that the results as presented in this chapter are correct, all written synthesized functions were used for automatic generation of bifurcation diagrams and Lyapunov exponents.
- **Simulation results.** Based on the results (Fig. 11.14 - Fig. 11.88) and the selected bifurcation diagrams, it can be stated that all simulations give satisfactory results and that evolutionary synthesis of chaos is capable of solving this class of problems.
- **Range of chaos and interval of observation.** During evolutions, chaos was searched by focusing on interval [0, 4], based on the a priori known behavior of the logistic equation, whose elements were used in the evolution. Despite the a priori known information, a few chaotic systems were located outside of this interval. That was due to the fact that a part of the chaotic behavior was inside the interval [0, 4] and thus EA was able to identify it. From these facts, it is clear that EA are able to locate chaos in a wider range than those expected from some textbook exemplary systems.
- **Exemplary system synthesis.** Based on the fact that the logistic equation (its elements and range) is used for chaos synthesis, it is logical to expect that during evolution (if repeated for many times) the original system should also be synthesized. That event was also observed for a few times, exactly in the mathematical form eq. (11.6).
- **Mutual comparison.** When comparing all algorithms, it is obvious that these algorithms produced good results. Parameter setting for the algorithms was based on a heuristic approach and thus there is a possibility that better settings can be found there. Based on these results, it is clear that for symbolic synthesis via analytic programming any evolutionary algorithm can be used.
- **Engineering design.** It is quite clear that evolutionary synthesis of chaos can be applied to engineering design of devices based on chaos (signal transmission via chaos, chaos-based encryption, and so on). Based on principles and results reported in this paper, it should be possible to synthesize systems with some precisely defined chaotic features and attributes.

Future research is being carried on under the framework of evolutionary synthesis of chaos. It is expected that all 13 EAs will also be used for synthesis of chaotic systems that are not restricted to the simple logistic equation. Based on the results reported in this chapter and our experience with EAs, it is believed that symbolic regression based on EAs is also able to synthesize various chaotic systems according to some predefined characteristics and conditions. It can be foreseeable that the possibility of synthesizing such artificial systems would have an impact on engineering applications dealing with chaos (signal transmission, cryptography, etc.), which is worth further investigation.

Acknowledgements. This work was supported by grant No. MSM 7088352101 of the Ministry of Education of the Czech Republic and by grants of the Grant Agency of the Czech Republic GACR 102/09/1680.

References

1. Alvarez, J., Puebla, H., Cervantes, I.: Stability of observer-based chaotic communitcation for a class of Lur'e systems. Int. J. Bifurcat Chaos Appl. Sci. Eng. 7, 1605–1618 (2002)
2. Back, T., Fogel, D., Michalewicz, Z.: Handbook of Evolutionary Computation, Institute of Physics, London (1997)
3. Beyer, H.: Theory of Evolution Strategies. Springer, New York (2001)
4. Cerny, V.: Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. J. Opt. Theory Appl. 45(1), 41–51 (1985)
5. Chen, G.: Controlling Chaos and Bifurcations in Engineering Systems. CRC Press, Boca Raton (2000)
6. Chen, G., Dong, X.: From Chaos to Order: Methodologies, Perspectives and Applications. World Scientific, Singapore (1998)
7. Chen, G., Ueta, T.: Yet another chaotic attractor. Int. J. Bifurcat Chaos Appl. Sci. Eng. 9, 1465–1667 (1999)
8. Davis, L.: Handbook of Genetic Algorithms. Van Nostrand Reinhold, Berlin (1996)
9. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. Wiley, Chichester (2001)
10. Dmitriev, A., Panas, A., Starkov, S.: Ring oscillating systems and their application to the synthesis of chaos generators. Int. J. Bifurcat Chaos Appl. Sci. Eng. 6(5), 851–865 (1996)
11. Dmitriev, A., Efremova, E., Kuzmin, L., Anagnostopoulos, A.: High dimensional RC – oscillators of chaos. In: International Symposium on Nonlinear Theory and its Applications, Miyagi, Japan (2001)
12. Eguchi, K., Inoue, T., Tsuneda, A.: Synthesis and analysis of a digital chaos circuit generating multiple-scroll strange attractors. IEICE Trans. Fundamentals E82-A(6), 965–972 (1999)
13. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
14. Gilmore, R., Lefranc, M.: The Topology of Chaos: Alice in Stretch and Squeezeland. Wiley Interscience, New York (2002)
15. Grebogi, C., Lai, Y.: Controlling chaos. In: Schuster, H. (ed.) Handbook of Chaos Control. Wiley-VCH, New York (1999)
16. Hilborn, R.: Chaos and Nonlinear Dynamics. Oxford University Press, UK (1994)
17. Holland, J.: Adaptation in Natural and Artificial Systems. Univ. Michigan Press, Ann Arbor (1975)
18. Hu, G., Xie, F., Xiao, J., Yang, J., Qu, Z.: Control of patterns and spatiotemporal chaos and its application. In: Schuster, H. (ed.) Handbook of Chaos Control. Wiley-VCH, New York (1999)
19. Johnson, C.: Artificial immune systems programming for symbolic regression. In: Ryan, C., Soule, T., Keijzer, M., Tsang, E.P.K., Poli, R., Costa, E. (eds.) EuroGP 2003. LNCS, vol. 2610, pp. 345–353. Springer, Heidelberg (2003)
20. Just, W.: Principles of time delayed feedback control. In: Schuster, H. (ed.) Handbook of Chaos Control. Wiley-VCH, New York (1999)

21. Just, W., Benner, H., Reibold, E.: Theoretical and experimental aspects of chaos control by time-delayed feedback. Chaos 13, 259–266 (2003)
22. Koza, J.: Genetic Programming: A paradigm for genetically breeding populations of computer programs to solve problems. Stanford University, Computer Science Department, Technical Report STAN-CS-90-1314 (1990)
23. Koza, J.: Genetic Programming. MIT Press, Boston (1998)
24. Koza, J., Keane, M., Streeter, M.: Evolving inventions. Scientific American, 40–47 (2003)
25. Koza, J., Bennet, F., Andre, D., Keane, M.: Genetic Programming III. Morgan Kaufnamm, New York (1999)
26. Lorenz, E.: Deterministic nonperiodic flow. Journal of the Atmospheric Sciences 20(2), 130–141 (1963)
27. May, R.: Simple mathematical model with very complicated dynamics. Nature 261, 45–67 (1976)
28. Nolle, L., Zelinka, I., Hopgood, A., Goodyear, A.: Comparison of an self organizing migration algorithm with simulated annealing and differential evolution for automated waveform tuning. Adv. Eng. Software 36(10), 645–653 (2005)
29. O'Neill, M., Ryan, C.: Grammatical Evolution. In: Evolutionary Automatic Programming in an Arbitrary Language. Springer, New York (2003)
30. O'Neill, M., Brabazon, A.: Grammatical Differential Evolution. In: Proc. International Conference on Artificial Intelligence (ICAI 2006), pp. 231–236. CSEA Press (2006)
31. Oplatkova, Z.: Optimal trajectory of robots using symbolic regression. In: Proc. 56th International Astronautics Congress 2005, Fukuoka, Japan, paper nr. IAC-05-C1.4.07 (2005)
32. Oplatkova, Z., Zelinka, I.: Investigation on artificial ant using analytic programming. In: Proc. Genetic and Evolutionary Computation Conference 2006, Seattle, WA, pp. 949–950 (2006)
33. Ott, E., Grebogi, C., Yorke, J.: Controlling chaos. Phys. Rev. Lett. 64, 1196–1199 (1990)
34. Perruquetti, W., Barbot, J.: Chaos in Automatic Control. CRC, Bota Raton (2005)
35. Price, K.: An Introduction to Differential Evolution. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 79–108. McGraw-Hill, London (1999)
36. Pyragas, K.: Continuous control of chaos by self-controlling feedback. Phys. Lett. A 170, 421–428 (1992)
37. Richter, H.: An evolutionary algorithm for controlling chaos: The use of multi-objective fitness functions. In: Guervós, J.J.M., Adamidis, P.A., Beyer, H.-G., Fernández-Villacañas, J.-L., Schwefel, H.-P. (eds.) PPSN 2002. LNCS, vol. 2439, pp. 308–317. Springer, Heidelberg (2002)
38. Richter, H., Reinschke, K.: Optimization of local control of chaos by an evolutionary algorithm. Physica D 144, 309–334 (2000)
39. Ryan, C., Collins, J., O'Neill, M.: Grammatical evolution: Evolving programs for an arbitrary language. In: Banzhaf, W., Poli, R., Schoenauer, M., Fogarty, T.C. (eds.) EuroGP 1998. LNCS, vol. 1391, p. 83. Springer, Heidelberg (1998)
40. Schuster, H.: Handbook of Chaos Control. Wiley-VCH, New York (1999)
41. Stewart, I.: The Lorenz attractor exists. Nature 406, 948–949 (2000)
42. Ueta, T., Chen, G.: Bifurcation analysis of Chen's attractor. Int. J. Bifurcat Chaos Appl. Sci. Eng. 10, 1917–1931 (2000)
43. Vanecek, A., Celikovsky, S.: Chaos synthesis via root locus. IEEE Trans. on Circ. and Systems 41, 59–60 (1994)
44. Vanecek, A., Celikovsky, S.: Control Systems: From Linear Analysis to Synthesis of Chaos. Prentice-Hall, London (1996)

45. Wang, X., Chen, G.: Chaotification via arbitrarily small feedback controls: Theory, method, and applications. Int. J. Bifurcat Chaos Appl. Sci. Eng. 10, 549–570 (2000)
46. Zelinka, I.: Analytic programming by Means of new evolutionary algorithms. In: Proc. 1st International Conference on New Trends in Physics 2001, Brno, Czech Republic, pp. 210–214 (2001)
47. Zelinka, I.: Analytic programming by means of soma algorithm. In: ICICIS 2002, First International Conference on Intelligent Computing and Information Systems, Cairo, Egypt, pp. 148–154 (2002)
48. Zelinka, I.: SOMA – Self Organizing Migrating Algorithm. In: Babu, B.V., Onwubolu, G. (eds.) New Optimization Techniques in Engineering, pp. 167–218. Springer, New York (2004)
49. Zelinka, I.: Investigation on evolutionary deterministic chaos control. In: Proc. IFAC, Prague, Czech Republic, paper No. 03187 (2005)
50. Zelinka, I.: Investigation on realtime deterministic chaos control by means of evolutionary algorithms. In: Proc. First IFAC Conference on Analysis and Control of Chaotic Systems, Reims, France, pp. 211–217 (2006)
51. Zelinka, I., Nolle, L.: Plasma reactor optimizing using differential evolution. In: Price, K., Lampinen, J., Storn, R. (eds.) Differential Evolution: A Practical Approach to Global Optimization, pp. 499–512. Springer, New York (2005)
52. Zelinka, I., Oplatkova, Z.: Analytic programming – Comparative study. In: Proc. the Second International Conference on Computational Intelligence, Robotics, and Autonomous Systems, Singapore, paper No. PS04-2-04 (2003)
53. Zelinka, I., Oplatkova, Z., Nolle, L.: Analytic programming – Symbolic regression by means of arbitrary evolutionary algorithms. Int. J. of Simulation, Systems, Science and Technology 6(9), 44–56 (2005)
54. Zelinka, I., Guanrong, C., Celikovsky, S.: Chaos Synthesis by Means of Evolutionary algorithms. Int. J. Bifurcat Chaos Appl. Sci. Eng. 18(4), 911–942 (2008)
55. Zhou, T., Chen, G., Celikovský, S.: An algorithm for computing heteroclinic orbits and its application to chaos synthesis in the generalized Lorenz system. In: Proc. 16th World Congress of the International Federation of Automatic Control [CD-ROM], Praha, Czech Republic (2005)
56. Zou, Y., Luo, X., Chen, G.: Pole placement method of controlling chaos in DC–DC buck converters. Chinese Phys. 15, 1719–1724 (2006)