Automata Network Dynamical Systems for Construction of Fractal Objects

V. M. Severyanov

Laboratory of Computing Techniques and Automation,
Joint Institute for Nuclear Research,
141980 Dubna, Russia

Abstract

The fractal geometry plays an important role in the contemporary science. In some sense, objects with integer dimension are partial cases of the more general realm of entities having ragged shape and fractional dimension. Fractals of a broad class are described by Deterministic Iterated Function Systems. Simultaneously the iterated function systems give a base for Automata Networks capable to realize their latent dynamics. When such a dynamics is becoming alive (with the help of an appropriate automata network), it finishes in a steady state which is (in the general case) a fractal set. In the report, an algorithm is described for building an automata network for a given iterated function system. It is worth to note that the automata networks can be considered as a generalization of cellular automata, the main difference is our automata networks have non-regular structure of the system of the cell neighborhoods. An evolving algebra approach for description of the automata network dynamical systems is also mentioned.

1 Introduction

Nature, as was apparent after B. Mandelbrot [1], has a predominantly fractal structure. *There comes a time of fractal geometry.* The fractal geometry plays an important role in the contemporary science. Its goal is to present a model for irregularity and fragmentation in Nature. Its realm can be traced from scales of galaxies up to subnuclear scales. Integer-dimensional objects are special cases of fractal-dimensional ones, the latter being most commonly encountered. This being the case, *fractal structures become an important part of the common language of science* [2], and we have to have some appropriate methods and tools to describe and manipulate them.

Fractals of a broad class are described by *Deterministic Iterated Function Systems* (IFSs) [2]. Beyond that point Deterministic IFSs enable us to describe recurrent asymmetric binary Neural Networks [3, 4] and a special class of Cellular Automata [5]. The last case is developed here further to the case of Automata Networks.

If an IFS is given, we can construct a *Neural Network* [3, 4, 8] or a *Cellular Automaton* [5] (think it as an Automata Network!) capable to realize its latent dynamics and generate an approximation of its attractor. The *collage theorem* [2] gives us hope to reach (in principle) any desired accuracy.

In the report, an algorithm is described for building an automata network for a given iterated function system. An evolving algebra approach for description of the automata network dynamical systems is also mentioned.

2 Iterated Function Systems

Following M. Barnsley [2], under *fractals* we mean *compact subsets* of a complete metric space.

A deterministic *Iterated Function System* (IFS) [2] is a structure

$$F = ((X,d), f_1, f_2, ..., f_N), \tag{1}$$

where (X,d) is a *complete* metric space, with metric d, $f_i : X \to X$ – a contiguous function $\forall i$. An IFS F induces an *operator* $F : \mathcal{H}(X) \to \mathcal{H}(X)$, where $\mathcal{H}(X)$ - the space of all compact subsets of X:

$$F = \bigcup_{i=1}^{q} f_i. \tag{2}$$

The metric space $(\mathcal{H}(X), h)$, with *Hausdorff metric* h, is also complete. When functions f_i are *contractions*, the IFS is *hyperbolic*. In this case, there is a compact subset A_F of X which is the fixed point of the operator F:

$$\exists A_F \in \mathcal{H}(X) : \; A_F = F(A_F). \tag{3}$$

Such a set is called the *attractor* of the IFS F. The pair $(\mathcal{H}(X), F)$ is a *set* dynamical system whose attractive point is a set (rather than a point) - the attractor A_F. Realizing the dynamics of such a dynamical system, we can build a fractal set.

The Hausdorff metric is defined as follows:

$$h(A,B) = \max(d_\mathrm{s}(A,B), d_s(B,A)), \qquad A, B \in \mathcal{H}(X), \tag{4}$$

where

$$d_\mathrm{s}(A,B) = \max_{x\in A} \min_{y \in B}(d(x,y)).$$

We are dealing here with hyperbolic IFSs with all f_i *affine* transformations.

3 Fractal Approximation of Sets (Barnsley's algorithm)

We can conveniently work in the unit square $S = [0,1] \times [0,1] \subset \mathbb{R}^2$ $S = [0,1] \times [0,1] \subset \mathbb{R}^2$ for the space X. The IFS under consideration is appropriately scaled to fit into S.

In practice, we are dealing with *approximations* of fractals rather than with fractals themselves. We work not in the space X but in its pixel representation $\tilde{X}$ and, therefore, we deal not with $(\mathcal{H}(X), h)$ but with $(\mathcal{H}(\tilde{X}), h)$ and, correspondingly, with $\tilde{F} = (\tilde{f}_1, \ldots, \tilde{f}_q)$ and $\tilde{S}$. In what follows under S, F and f we mean $\tilde{S}$, $\tilde{F}$ and $\tilde{f}$, respectively.

One way to build a fractal [4], specified by IFS, looks as follows. We take an initial $A_0 \in \mathcal{H}(X)$ and define

$$A_{n+1} = F(A_n) \equiv \bigcup_{i=1}^{q} f_i(A_n), \qquad n = 0, \ldots, \infty. \tag{5}$$

The sequence $\{A_n\}_{n=0}^{\infty}$ converges to A_F in the Hausdorff metric. We do N iterations of F. When the number N of iterations becomes great enough, we have an approximation

$$F^N(A_0) \approx A_F. \tag{6}$$

Barnsley's deterministic algorithm [2] for building a fractal implies the calculation of each $f_k \in F$ for every point $s_i \in S$, $i \in \{1, \ldots, |S|\}$ for every iteration. When we wish to rebuild the fractal, we have to repeat the whole above procedure.

4 Neural Network Algorithms

4.1 Neural Networks

The base element of a neural network model, a *Formal Neuron* [6], computes

$$y(\mathbf{x}, \mathbf{w}) = g(\mathbf{x} \cdot \mathbf{w}) \tag{7}$$

the inner product of its *input vector* $\mathbf{x} = (x_0, x_1, \ldots, x_n)^\top$ and its *synaptic weight vector* $\mathbf{w} = (w_0, w_1, \ldots, w_n)^\top$ which then goes through *activation function* g to yield neuron's *output* y. The input $x_0 = 1$, and the corresponding synaptic weight w_0 is called the *threshold* of the neuron. As activation function g, either Heaviside (step) function or a sigmoid function is commonly used.

A *Neural Network Model* [7] is an interconnection of neurons. IFSs describe *recurrent* models. To be more precise, they describe *binary recurrent asymmetric* neural networks.

4.2 Stark's Neural Network Algorithm

Stark's algorithm [3, 4] implies the transition from a given IFS F to the corresponding neural network $\mathbf{W}$. To do such a passage, it is enough to consider any one iteration of F.

Let $X \equiv S^u \subset \mathrm{I\!R}^2$, S^u – a unit square digitized with respect to fixed resolutions Δ_x and Δ_y. Every A_n has its indicator function, $y : S^u \to \{0, 1\}$, such that:

$$\forall s \in S^u : \quad y_s(n) = \begin{cases} 1 & \text{if } s \in A_n \\ 0 & \text{if } s \notin A_n \end{cases} \tag{8}$$

If we define

$$w_{ss\prime} = \begin{cases} 1 & \text{if } f_i(s\prime) = s \quad \text{for some } i \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

then the dynamics of $y_s(n)$ is given by

$$y_s(n+1) = g\Big(\sum_{s\prime \in S} w_{ss\prime} y_{s\prime}(n)\Big), \tag{10}$$

where g is a step function.

Thus, we have the binary neural network with $|S^u|$ neurons y_s and synaptic weights $w_{ss\prime}$ that realizes the IFS F.

4.3 Our Neural Network Algorithm

The main idea of our algorithm [8] is based on the known feature of affine transformations: transformation of a given rectangle is completely conditioned by transformation of three of its corner points.

As previously, we work inside a unit square S^u of pixels for which we select three corner points (s_0, s_t, s_r), s_0 being left lower corner, s_t left top corner, s_r right lower corner. For every $f_k \in F$ $(k = 1, \ldots, |F|)$ we only compute images of these three corner points, $(s'_0 = f_k(s_0), s'_t = f_k(s_t), s'_r = f_k(s_r))$. Then we define offsets $\Delta^t_x = x(s'_t) - x(s'_0)$, $\Delta^t_y = y(s'_t) - y(s'_0)$, $\Delta^r_x = x(s'_r) - x(s'_0)$, and $\Delta^r_y = y(s'_r) - y(s'_0)$ and use them in computing $\tan\alpha$ and $\tan\beta$, where α – the angle with x axis and β – the angle with y axis. This computation gives us the set $f_k^{-1}(s'')$ of points which are transformed by f_k onto s'' and, therefore, the synaptic weights $w_{s''s} = 1$. Horizontal size $h(f_k^{-1}(s''))$ and vertical size $v(f_k^{-1}(s''))$ of the rectangle $f_k^{-1}(s'')$ are determined by the contraction factors of f_k on x and y axes and calculated only once for each f_k. The inclination of each side of the rectangle $f_k^{-1}(s'')$ is taken into account by $\tan\alpha$ and $\tan\beta$, respectively.

The dynamics of the neural network is the same as in the Stark's algorithm.

5 Automata Network Algorithm

5.1 Automata Networks

We define an *Automata Network* σ as a structure

$$(C, A, S_0, N, \Delta), \tag{11}$$

where

- C is a set of *cells*;

- A is an *alphabet of states* (attributes);
- $S : C \to A$ is a *state*, S_0 - an *initial* state;
- $N : C \to \mathcal{P}(\mathcal{C})$ is a *neighborhood system*, $\mathcal{P}(\mathcal{C})$ is the power-set of C, $\forall c \in C : \; N(c)$ is the *neighborhood* of c;
- $\Delta : \Sigma \to \Sigma$ is a *global dynamic rule* (GDR), $\Sigma = \{S \,|\, S \text{ is a state}\}$.

$$\Delta^n : S_{n-1} \longmapsto S_n \qquad n = 1, 2, \ldots, M \tag{12}$$

The global dynamic rule Δ comes about from the *local dynamic rules* (LDRs)

$$\delta_c : \Sigma_{N(c)} \to A, \tag{13}$$

where $S_{N(c)}$ is the restriction of S to $N(c)$. Every LDR δ_c gives a new state value to the cell c as a function of cell states from the neighborhood $N(c)$ of c. The action of Δ

$$\Delta(S) = \bigcup_{c \in C} S[S(c) \leftarrow \delta_c(S_{N(c)})], \tag{14}$$

is the union of the states obtained by replacing the state of each cell c accordingly to δ_c.

Automata Networks can be considered as a generalization of Cellular Automata, the main difference is our automata networks have non-regular structure of the system of the cell neighborhoods. There is no a common template of vicinity for cells, the neighborhoods of cells have variable cardinality and, hence, the LDRs have variable arity. There are possible cases when $c \notin N(c)$ and even $N(c) = \{\emptyset\}$.

An automata network σ works as follows. If an initial state S_0 is given, it iteratively applies the GDR Δ until it reaches a steady attractive state S_A:

$$S_0 \xrightarrow{\Delta} S_1 \xrightarrow{\Delta} S_2 \xrightarrow{\Delta} \ldots \xrightarrow{\Delta} S_A.$$

The cells work synchronously in parallel governed by some global synchronization signals.

5.2 Iterated Function Systems as Automata Networks

When we want to realize the dynamics of a *digitized* IFS

$$F = ((X, d), f_1, f_2, ..., f_k)$$

with the help of an automata network σ, we have the following picture.

As C we take a regular lattice of some dimensionality m. In a practically interesting case, $m = 2$ and $C = \{(i, j) \,|\, i = 1, \ldots, p,\ j = 1, \ldots, q\}$. The alphabet of states $A = \{0, 1\}$. The cell neighborhoods are defined by the inverses of the IFS's functions:

$$\forall c \in C : \quad N(c) = \bigcup_{i=1}^{k} f_i^{-1}(c). \tag{15}$$

The LDRs δ_c are the same for all cells c:

$$\forall c \in C: \quad \delta_c(S_{N(c)}) = \begin{cases} 1 & \text{if } \exists c' \in N(c) \;\; S(c') = 1 \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

To build a bridge to our neural network algorithm (Sec. 4.3), we take as C our unit square S^u of pixels and obtain the neighborhoods $N(c)$ by analogy with computing the synaptic weights.

The whole process of going from an *initial* IFS

$$F = ((X,d), f_1, f_2, ..., f_k) \qquad \& \qquad F = \bigcup_{i=1}^{k} f_i$$

to the corresponding automata network σ looks like the following.

Fist of all we go to the dynamical system

$$((\mathcal{H}(X), h), F),$$

then, after digitization and scaling, to the dynamical system

$$((\mathcal{H}(S^u), h), F),$$

where S^u is the unit square.

Then we use our Automata Network Algorithm to get a special intermediate form of the IFS F which we call *"geometrical form"*:

$$F_g = \{< s_0^i, s_t^i, s_r^i >\}_{i \in \{1,2,\ldots,k\}}$$

where s_0^i, s_t^i, s_r^i are the images of the corner points s_0, s_t, s_r under $f_i \in F$ (Sec. 4.3).

We continue to apply the Automata Network Algorithm and eventually construct the automata network

$$(S^u, \{0,1\}, N, A_0, \Delta) \qquad \& \qquad N = \bigcup_{s \in S^u} \bigcup_{f \in F} f^{-1}(s).$$

We start the dynamical process, beginning from a compact set $A_0 \subseteq S^u$, and after a big enough number M of iterations get an approximation A_M of the IFS's attractor A_F:

$$\Delta^* : A_0 \longrightarrow A_M \approx A_F.$$

6 Evolving Algebras

Evolving Algebras have been proposed by Yuri Gurevich in 1988 [9, 10] as models for arbitrary computational processes. They are finite, dynamic algebras representing state transitions and describing operational semantics of discrete dynamical systems. They may be tailored to any desired level of abstraction. System states are represented here as static algebras.

An evolving algebra is a structure

$$\psi = (\Sigma, S, I_0, T),$$

where:

- Σ is a signature (a set of operational (or functional) symbols with their arities);
- S is a superuniverse, $\{true, false\} \subset S$;
- I_0 - an initial interpretation (a finite, static, many-sorted algebra);
- T - a set of transition rules.

I_0 gives an initial interpretation of the signature's operational symbols:

$$I_0 : \Sigma \to \bigcup_{n \geq 0} (S^n \to S) \qquad \forall f \in \Sigma \quad I_0(f) : S^n \to S.$$

There are four kinds of transition rules (or updates):

1. **Function updates**

$$f(t_1, \ldots, t_n) := t_0 \qquad f \in \Sigma, \ \ n \geq 0$$

2. **Conditional**

$$if\ b\ then\ C$$

3. **Extension of universes**

$$extend\ U \subset S\ by\ x_1, \ldots, x_m$$

4. **Contraction of universes**

$$discard\ t\ from\ U$$

Iterative application of evolving algebra ψ to sequentially arising states (static algebras) I_i, starting from the initial state I_0, may give a terminated computation:

$$I_0 \xrightarrow{\psi} I_1 \xrightarrow{\psi} I_2 \xrightarrow{\psi} \ldots \xrightarrow{\psi} I_M.$$

We plan to use the very flexible and powerful formalism of evolving algebras for description of our above-mentioned algorithms.

7 Conclusion

We have considered here an outline of our Automata Network Algorithm for construction of fractal objects, The algorithm is a natural evolution of our previous Neural Network Algorithm but the former is simpler and more efficient because there is no need to calculate and store synaptic weights. So we have the possibilities to calculate fractal attractors with the help of automata networks or by neural computations. As very interesting perspective we mentioned the intention to apply the formalism of Evolving Algebras for describing our algorithm. This is a subject for some future investigations.

References

[1] Mandelbrot, B.: The Fractal Geometry of Nature. W. H. Freeman and Co., San Francisco, 1982

[2] Barnsley, M. F.: Fractals Everywhere. Springer-Verlag, Berlin, 1986

[3] Stark,J.: Neural Networks **4** (1991) 679

[4] Bressloff, P. C., and Stark, J.: Neural Networks, Learning Automata and Iterated Function Systems. In: Fractals and Chaos, ed. A. J. Crilly, R. A. Earnshow, H. Jones et al., Springer-Verlag, New-York, 1991, 145

[5] Severyanov, V. M.: Parallel Computation of Fractal Sets with the Help of Neural Networks and Cellular Automata. In: Parallel Computing Technologies, Proc. IV Int. Conference, PACT-97, Yaroslavl, Russia, September 1997, ed. B. Victor Malyshkin, Springer-Verlag, Berlin, 1997, 109

[6] Rumelhart, D. E., Hinton, G. E., and Williams, R. J.: Parallel Distributed Processing. MIT Press, Cambridge, 1986

[7] Humpert, B.: Comp. Phys. Comm. **58** (1990) 223

[8] Severyanov, V. M.: Calculation and Interactive Construction of Flat Fractals Using Neural Networks. In: New Computing Techniques in Physics Research IV, Proc. IV Int. Workshop on Software Engineering and Artificial Intelligence for High Energy and Nuclear Physics, Pisa, Italy, 3-8 April, 1995, ed. B. Denby, D. Perret-Gallix, Word Scientific, Singapore, 1995, 509

[9] Yuri Gurevich: Logic and challenge of computer science. In E. Börger, editor, *Specification and Validation Methods.* Oxford University Press, 1995

[10] Yuri Gurevich: Evolving Algebras: a tutorial introduction. *Bulletin of the European Association for Theoretic Computer Science*, 43:264-284, 1991