Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Physica D 181 (2003) 274–285
PHYSICA D
www.elsevier.com/locate/physd

Chaos automata: iterated function systems with memory

Dan Ashlock [a,*], Jim Golden [b]

[a] *Mathematics Department and Bioinformatics and Computational Biology Program, Iowa State University, Ames, IA 50011, USA*
[b] *CuraGen Corporation, 555 Long Wharf Drive, 9th Floor, New Haven, CT 06511, USA*

Received 22 August 2001; received in revised form 10 August 2002; accepted 10 April 2003
Communicated by A.C. Scott

Abstract

Transforming biological sequences into fractals in order to visualize them is a long standing technique, in the form of the traditional four-cornered chaos game. In this paper we give a generalization of the standard chaos game visualization for DNA sequences. It incorporates iterated function systems that are called under the control of a finite state automaton, yielding a DNA to fractal transformation system with memory. We term these fractal visualizers *chaos automata*. The use of memory enables association of widely separated sequence events in the drawing of the fractal, finessing the "forgetfulness" of other fractal visualization methods. We use a genetic algorithm to train chaos automata to distinguish introns and exons in *Zea mays* (corn). A substantial issue treated here is the creation of a fitness function that leads to good visual separation of distinct data types.

PACS: 87.15.Aa; 87.15.Cc

Keywords: Biological fractals; Chaos games; Evolutionary computation

1. Introduction

This paper is the second [1] in a series connecting three disparate threads in the computational interpretation of biological data through visualization. The first of the threads, chaos game fractals or iterated function systems [4,7], seek to give useful visual displays of large, possibly complex data sets [6,9,11,14,17]. The second, bioinformatics [3,16], has as one of its goals the automation of analysis and interpretation of large biological data sets. The third, evolutionary computation [2], is used to connect the first two. Evolutionary computation is a term used to describe the field including genetic algorithms [8,10], evolutionary programming, [15], and genetic programming [12,13]. Briefly, evolutionary computation is a technique for locating structures that satisfy some pre-specified figure of merit via a digital instance of the theory of evolution. In this research, evolutionary computation is employed to produce data-driven iterated function systems whose fractal attractors form a visual presentation of information contained in sequences of DNA. The training via evolutionary algorithm permits us to create fractals that highlight some biological feature of interest. In the research presented here we visually distinguish introns and exons. The sequences of DNA being displayed replace the random numbers that usually drive such fractal processes. The problem we solved is that of producing visual summaries of large pieces of DNA data that highlight selected differences

* Corresponding author.
E-mail address: danwell@iastate.edu (D. Ashlock).
URL: http://www.curagen.com.

0167-2789/03/$ – see front matter
doi:10.1016/S0167-2789(03)00122-2

of interest, in this case distinguishing introns and exons.

The innovation presented is a new way of coding iterated function system fractals called *chaos automata*. Chaos automata differ from standard iterated function systems in that they retain internal state information and so gain the ability to visually associate events that are not proximate in their generation histories. Two more-or-less similar sequences separated by a unique marker could, for example, produce very different chaos automata based fractals by having the finite state transitions recognize the marker and then use different states, and hence collections of contraction maps, on the remaining data. This sort of long term memory is not present in any other sort of iterated function system visualization system presented to date. The problem addressed by the use of internal state information is that when data or random numbers are used to drive a standard iterated function system [4] the lack of internal state information means that data items are "forgotten" as their influence vanished into the contractions of space associated with each map in such a system.

2. Detailed specification of chaos automata

We draw the contraction maps for the current version of chaos automata from a class of maps, similitudes, that are guaranteed to be contraction maps. A *similitude* is a map that performs a rigid rotation of the plane, displaces the plane by a fixed amount, and then contracts the plane toward the origin by a fixed scaling factor. The derivation of a new point $(x_{\text{new}}, y_{\text{new}})$ from old point (x, y) with a similitude that uses rotation t, displacement $(\Delta X, \Delta Y)$ and scaling factor $0 < s < 1$ is given by

$$x_{\text{new}} = s(x\cos(t) - y\sin(t) + \Delta x), \tag{1}$$

$$y_{\text{new}} = s(x\sin(t) + y\cos(t) + \Delta y). \tag{2}$$

To see that a similitude must always reduce the distance between two points, note that rotation and displacement are isometries, they do not change distances between points. This means any change is due to the scaling factor which necessarily causes a reduction in the distance between pairs of points.

We can now give a formal definition of chaos automata. A chaos automata is a 5-tuple (S, C, A, t, i) where S is a set of states, C a set of contraction maps from $\mathbb{R}^2$ to itself, A an input alphabet, $t : S \times A \to A \times C$ a transition function that maps a state and input to a next state and contraction map, and $i \in S$ a distinguished initial state. To generate a fractal from a stream of data with a chaos automata we use the algorithm given in Fig. 1. Readers familiar with finite state machines will note we have made the, somewhat arbitrary, choice of associating our contraction maps with states rather than transitions. An example of a chaos automata, evolved to be driven with DNA data, is showing in Fig. 2.

```
Set state to initial state i.
Set (x,y) to (0,0).
Repeat
    Apply the contraction map for the current state to (x,y).
    Plot(x,y).
    Update the state by applying t to the current state and input.
Until (out of input).
```

Fig. 1. Basic algorithm for generating fractals with a chaos automata.

```
Starting State: 3

State  If C  G  A  T

---------------------  Functions:

 0)       0  0  0  3  : R:0.156 D:( 0.352,-0.858) S:0.540

 1)       0  2  1  6  : R:1.722 D:( 0.710, 1.493) S:0.658

 2)       6  3  2  2  : R:0.150 D:( 0.700, 0.913) S:0.763

 3)       6  2  2  2  : R:0.105 D:( 0.709, 1.375) S:0.689

 4)       3  2  2  6  : R:5.761 D:( 0.331, 1.721) S:0.804

 5)       2  1  2  4  : R:5.993 D:( 0.015, 0.722) S:0.896

 6)       0  2  0  2  : R:1.044 D:( 1.184,-0.374) S:0.732

 7)       3  4  4  3  : R:4.732 D:( 0.753, 0.750) S:0.664
```

Fig. 2. A chaos automata evolved to visually separate intronic and exonic DNA. The automata starts in state 3 and makes state transitions depending on inputs from the alphabet C, G, A, T. As the automata enters a given state it applies the similitude defined by a rotation (R), displacement (D), and shrinkage (S). These similitudes operate on a moving point to generate fractal images in the standard manner of an iterated function system.

3. The data structure and its variation operators

In order to use an evolutionary algorithm to evolve chaos automata that visually separate classes of data, they must be implemented as a data structure endowed with variation operators (cross-over and mutation operators). The data structure consists of an integer, storing the identity of the initial state, together with a vector of *nodes*, each of which holds the information defining one state. This information consists of four integers, giving the next state to use for inputs of C, G, A, and T, respectively, and a similitude to apply when the automata makes a transition to the state. The similitude in turn contains four real parameters, two coordinates for planar displacement in the range $-1 < \Delta X, \Delta Y < 1$, the angle of rotation $0 \leq t \leq 2\pi$, and a contraction factor in the range $0 < s < 1$.

A two point cross-over operator is used. This cross-over operator treats the vector of nodes as a linear chromosome. Each node is an indivisible object to the cross-over operator and so the cross-over is essentially the classical two-point cross-over, relative to these indivisible objects. The integer that identifies the initial state is attached to the first node in the chromosome and moves with it during cross-over.

There are three kinds of things that could be changed with a "mutation" operator. Primitive mutation operators are defined for each of these things and then used in turn to define a master mutation operator that calls the primitive mutations with a fixed probability schedule. The first primitive mutation acts on the initial state, picking a new initial state uniformly at random. The second primitive mutation acts on transition to a next state and selects a new next state uniformly at random. The third primitive mutation is similitude mutation. Similitude mutation picks one of the four numbers defining the similitude, ΔX, ΔY, t or s, uniformly at random and adds a uniformly distributed number in the range $-0.1 < R < 0.1$ to it. If either of the displacements becomes larger than 1 in absolute value it is reflected back into the range $-1 < \Delta X, \Delta Y < 1$. The shrink value s is likewise

reflected off of the boundaries of its range $0 < s < 1$ when it wanders out. The angular parameter, being periodic, does not require boundary correction. The master mutation mutates the initial state 10% of the time, a transition 50% of the time, and mutates a similitude 40% of the time. These percentages were chosen in an ad hoc manner.

4. Evolution and fitness

The goal of this research is to visually display the difference between distinct types of DNA data. We use an evolutionary algorithm to locate chaos automata that have this property of visually separating data in the fractal generated by the chaos automata. As so often happens in evolutionary computation the difficult question, once the data structure or "chromosome" has been chosen, is that of choosing the fitness function. If the only goal were to computationally separate two or more classes of data then a simple assessment of the fraction of times the evolving structure separated the data classes of interest would be enough. To *visually* separate data a more sophisticated fitness function is required. In preliminary studies, separating the mean position of points plotted for two distinct data types resulted in two dots being plotted. This solution separates the classes of data most thoroughly, but provide only the most minimal of visual cues. This motivated the examination of several fitness functions as well as the imposition of a lower bound on the contraction factor of the similitudes.

The meaning of the position of points plotted by a chaos automata is not known a priori but is substantially influenced by the fitness function. Multiple simulations will generate chaos automata, for the same set of training data, that produce radically different sets of plotted points. In this, chaos automata differ substantially from the chaos games that they generalize. The use of internal state information permits chaos automata to visually display phenomena that standard chaos games could not but the meaning of those points must be deduced by comparison with the data. This can be done, for example, by associating different colors with different types of data.

To describe the fitness functions we employ a slight abuse of notation: the moving point, used to generate fractals from chaos automata driven by data, are referred to as if its coordinates were a pair of random variables. Thus (X, Y) are an ordered pair of random variables that give the position of the moving point of the chaos game. When working to separate several types of data $\{d_1, d_2, \ldots, d_n\}$ the points described by (X, Y) are partitioned into $\{(X_{d_1}, Y_{d_1}), (X_{d_2}, Y_{d_2}), \ldots, (X_{d_n}, Y_{d_n})\}$, which are the positions of the moving points when a chaos automata was being driven by data of types $d_1, d_2, \ldots, d_n$, respectively. For any random variable R we use $\mu(R)$ and $\sigma^2(R)$ for the sample mean and variance of R.

As an iterated function system driven by random numbers is applied to a moving point the point approaches a fractal attractor [4]. If the fractal is not known a priori then the iterated function system must be "burned-in" or run for many time steps to ensure that the point starts in or near the system's attractor. Throughout this research a three-part burn-in, diagrammed in Fig. 3, is employed. The moving point is initialized to lie at the origin of $\mathbb{R}^2$. In the first part of burn-in the chaos automata is driven with data selected uniformly at random from {C, G, A, T} for 1000 iterations. This places the moving point in or near the attractor of the chaos automaton for uniform data. In the second portion of burn-in the chaos automata is

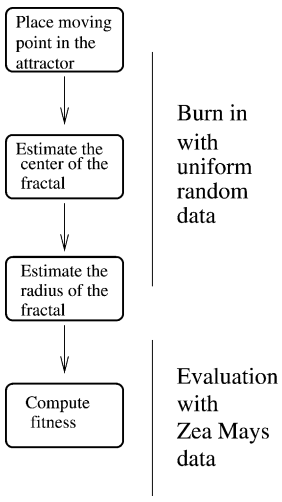

Fig. 3. Steps involved in fitness evaluation.

driven with uniform random data for 1000 additional steps, generating 1000 sample values of (X, Y). The values of $\mu(X)$ and $\mu(Y)$ during this phase of burn-in are saved for later use in normalizing both fitness and any images generated. In the third part of burn-in the chaos automata is driven for an additional 1000 steps. Over those 1000 samples of (X, Y) the maximum Euclidean distance $R_{\max}$ of any instance of the moving point from $(\mu(X), \mu(Y))$ is computed. We term $R_{\max}$ the *radius* of the fractal associated with the chaos automata. With these definitions in hand it is possible to succinctly give the four fitness functions used in this research.

The first

$$F_1 = \sqrt{(\mu(X_{d_1}) - \mu(X_{d_2}))^2 + (\mu(Y_{d_1}) - \mu(Y_{d_2}))^2} \tag{3}$$

computes the Euclidean distance between the mean position of the moving points for each of two data types. The second fitness function

$$F_2 = \sum_{(x,y)\in(X_{d_1},Y_{d_1})} g(x, y) + \sum_{(x,y)\in(X_{d_2},Y_{d_2})} g(x-1, y-1), \tag{4}$$

where

$$g(x, y) = \frac{1}{4x^4 + 4y^4 + 1} \tag{5}$$

places a steep sided, relatively flat-topped "hill", shown in Fig. 4 with its mode at $(0, 0)$ and a second hill with its mode at $(1, 1)$. Fitness is increased by the height of the first hill at (x, y) if an instance of the moving point (x, y) was the result of driving the chaos automata with data of the first type and by the height of the second hill at (x, y) if that moving point resulted from data of the second type.

The third fitness function

$$F_3 = \sigma(X_{d_1})\sigma(Y_{d_1})\sigma(X_{d_2})\sigma(Y_{d_2})F_1 \tag{6}$$

is a modification of the first fitness function that scales the fitness by the scatter of the moving points within each data class. As we will see in Section 5, this placed to much emphasis on scatter and led us to try a fourth fitness function:

$$F_4 = \tan^{-1}(\sigma(X_{d_1})\sigma(Y_{d_1})\sigma(X_{d_2})\sigma(Y_{d_2}))F_1, \tag{7}$$

which rewards additional scatter only when scatter is small by running the standard-deviation based term through a bounded increasing function, the arctangent.

4.1. The evolutionary algorithm

The evolutionary algorithm used in this research is a *steady-state* algorithm [18]. Rather than having discrete generations, mating events happen within the population in a steady-state fashion. In each mating

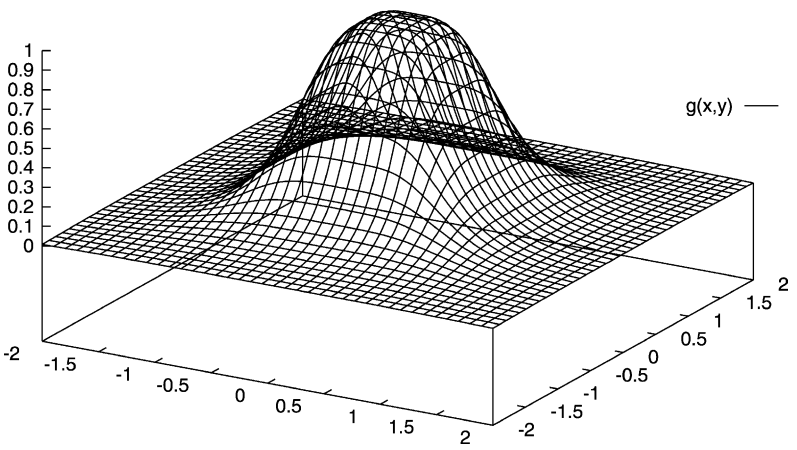

Fig. 4. The function $g(x, y)$ used to envelope fitness values for the fitness function F_2.

event four chromosomes are chosen uniformly at random, without replacement, from the population. The two most fit are then crossed over and the resulting new chromosomes replace the two least fit. Each of the new individuals is subjected to a single mutation.

In earlier work generating fractals from DNA data [1] initial trials were performed with several different types of evolutionary algorithm: steady state, generational with tournament selection, generational with rank selection. The results were found to be insensitive to the choice of algorithm type. The use of a steady-state algorithm is believed to be a less-than-critical feature of this work.

4.2. Experimental design

We performed a series of evolutionary simulations. The first set of simulations did not use biologically derived DNA and were intended to characterize the behavior of the four fitness functions and their interaction with the chaos automata data structure. These simulations used two types of synthetic, random DNA. The chaos automata/evolutionary algorithm system was subsequently tested on its ability to separate distinct types of data drawn from biological sources. These simulations used intron and exon sequence drawn from *Zea mays* (corn) genes deposited in GenBank. For each fitness evaluation a given chaos automata was run, after the burn-in phase on uniform random data, through 5000 bases of DNA, alternating runs of exon and intron DNA. All records from GenBank for the organism *Zea mays* (corn) that contained the features "Intron", "Exon", or "complete CDS" were down-loaded. Where possible, introns and exons were excised from these records. After processing, 610 exons and 497 introns were rescued. GenBank records where there were not one more exon than intron, which had predicted rather than experimentally verified features, or were difficult to interpret, were discarded.

Fitness evaluations was performed by feeding the chaos automata whole introns and exons successively until the total number of DNA bases used exceeded 5000 bases. The available introns and exons were used in continuous rotation during the course of evolution. Two sets of simulations with introns and exons were performed using the same evolutionary algorithm and population parameters as for the two random source data. The first used all available intron and exon data for fitness evaluation (training) of the chaos automata. The second set of simulations were performed with 20 introns and exons reserved from the training data for later use in cross-validation.

In all the simulations a population of 120 8-state chaos automata were evolved for 20,000 mating events, saving the most fit chaos automata from the final generation in each simulation. The populations were initialized by filling in the chaos automata with appropriate uniform random values. Each type of simulation, defined by a data type and fitness function choice, was run 50 times with distinct random populations. The two random source data consisted of a stream of bases drawn uniformly at random from {C, G, A} for the first source and a stream of bases drawn uniformly at random from {G, A, T} for the second.

Initial sets of simulations exposed two potential flaws in the system. First of all, since the position of moving points is normalized by the radius, $R_{\max}$, of the fractal it is possible to obtain incremental fitness by reducing the radius of the fractal. Because of this the average radius of population members rapidly dropped to 10^{-15} in all simulations, which made the fractals difficult to view. This flaw was removed by fiat: any fractal with a radius of 0.1 or less was awarded a fitness of zero. The number 0.1 was chosen because it was smaller than the radius of the substantial majority of a sample of 1000 randomly generates chaos automata driven with uniform, random data for 5000 bases.

The second flaw was relative to the desire to have visually appealing fractals. If the contraction factor of the similitudes in a chaos automata are close to zero then the automata displays only a sparse dusting of points. Such a sparse dust obtains a high fitness as it does spatially separate the points plotted for each data type. Under fitness functions F_1 and F_2, for example, a chaos automata can get high fitness by simply placing the moving point on a single pixel for each of two data classes. To amend this flaw we placed a lower bound on the contraction factor s with the reflection used to correct mutations that drifted below zero being

replaced with reflection at the lower bound. This lower bound was either 0.8 (for simulations performed with F_1 and F_2) or 0.5 (for simulations performed with F_3 and F_4). These values were chosen in an ad hoc manner.

5. Results

In order to demonstrate that the chaos automata selected by an evolutionary algorithm can separate data classes the system was first tested for all four fitness function on the two random source data. Representative fractals for each fitness function are shown in Fig. 6. All the simulations had a substantial increase in the population maximum of the fitness, as compared to the fitnesses in the initial random population, over the course of evolution. This increase spanned more than an order of magnitude. Both between fitness functions and between distinct simulations for each fitness function the major difference was in the degree to which the separation of the data classes was visually striking. An example of a visually striking separation is given in Fig. 5. We note that the color versions of these fractals are more impressive, but the visual separation is still present in black-and-white or gray-scale images.

5.1. Utility of particular fitness functions

A representative best-of-simulation fractal from each set of 50 simulations run on two random source training data for each fitness function is shown in Fig. 6. Fitness function F_1 produced large, visually appealing fractals but suffered from a problem encountered in a similar experiment that used a lookup table in place of a chaos automata [1]: often the portions of the fractal associated with each of the two data classes were closely intertwined. A few fractals, including the representative shown in Fig. 6, had some spatial separation.

Fitness function F_2 caused the chaos automata to very cleanly separate the two data types but, even with a large lower bound on the contraction factor s in each similitude, still has a strong tendency to place the moving point into a very small attractor for each data type. The representative sample shown is one of

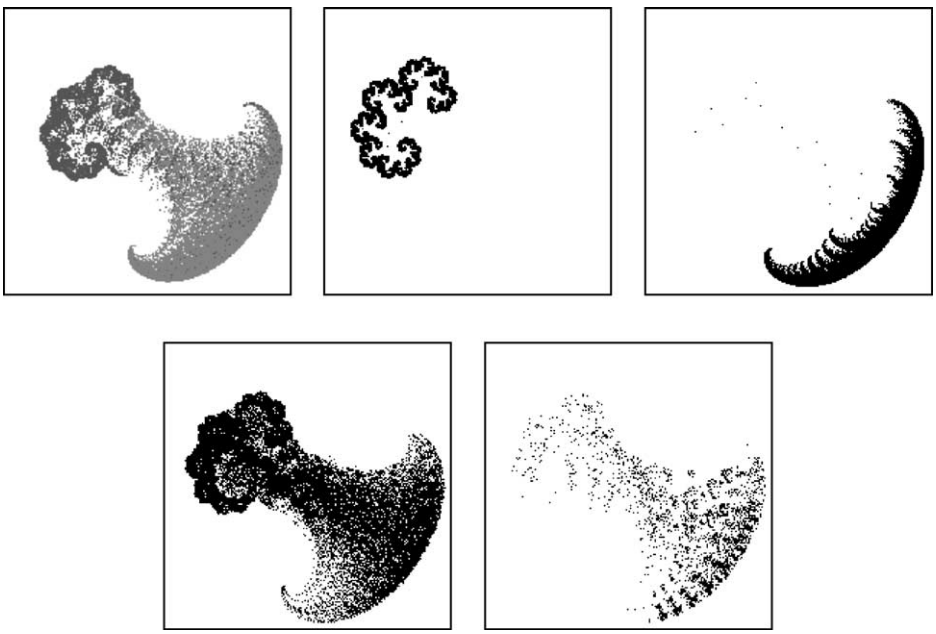

Fig. 5. The result of simulation number 19 using fitness function F_4 on maize intron and exon data with a lower bound of 0.5 on the contraction factor s. Shown are the full fractal, the fractal driven with each of the two distinct random sources, and the fractal driven with the intron and exon data, in reading order.

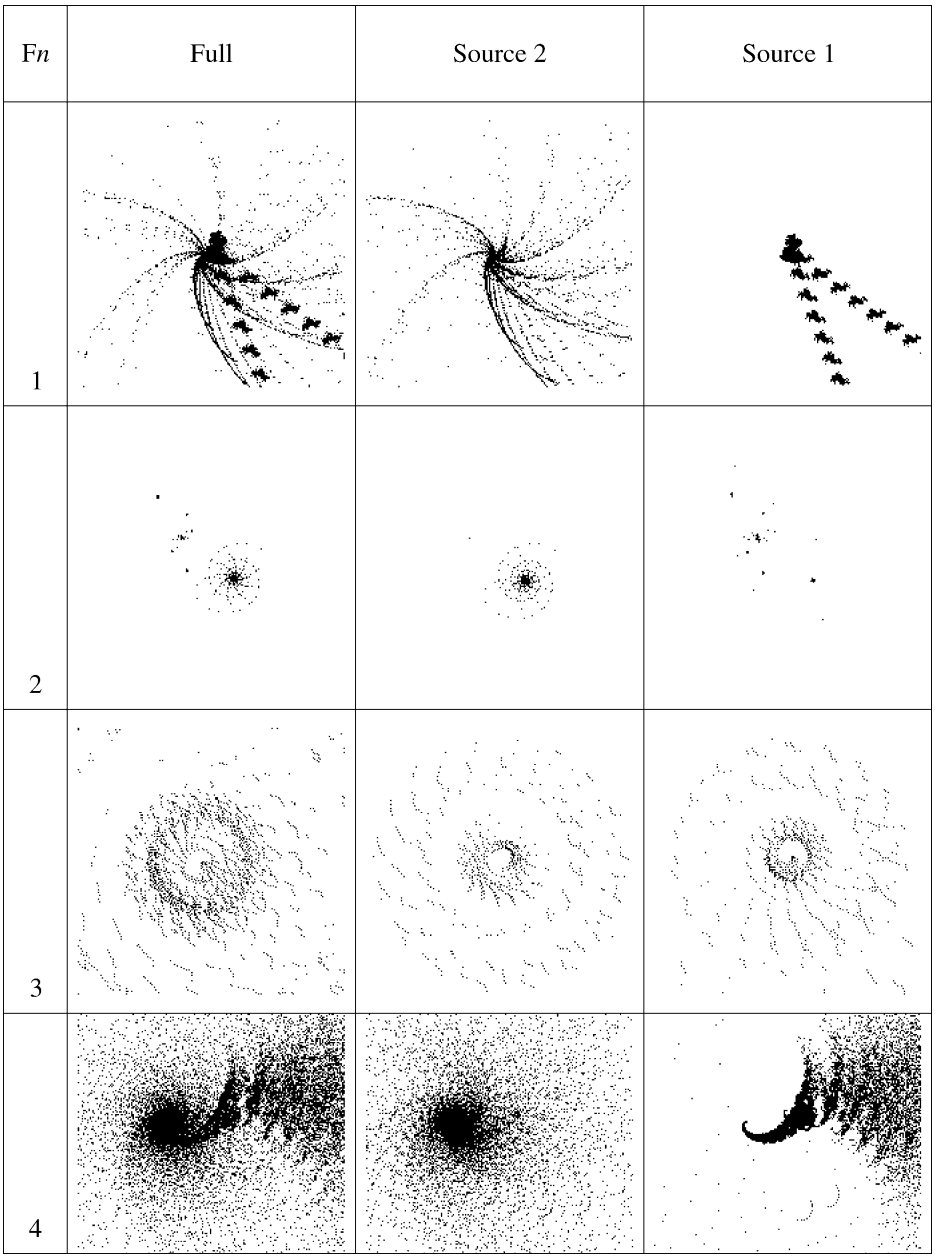

Fig. 6. A sample of the fractals produced for each fitness function on the random source data. The full fractal and the result of driving it with both types of random source independently are shown.

the *least* compact results of using fitness function F_2. It may be possible to get better results by moving the position of, or modifying the shape of, the two "mesas" that permit the fitness function to reward positional differences.

Fitness function F_3, which had an unbounded reward for scatter of the moving point, produced very large scatter. Typically the fractals resulting under F_3 would have successive regions of the plane, in concentric waves, associated with each of the two random

sources. The representative fractal drawn from this set of simulations resembles the pattern resulting from two adjacent drops of water in a pond, one drop associated with each of the two data sources.

Fitness function F_4, which had a bounded reward for scatter of the moving point, produced many fractals which both spatially separated the two data types and managed to scatter enough to be visually appealing. One of the better separators from the set of simulations using F_4 was chosen as the representative. The fractal shown in Fig. 5 also uses fitness function F_4 but is evolved using biological rather than random source data. Because of its superior performance all subsequent work uses F_4.

5.2. Intron and exon data from Zea mays

The fractals in Fig. 5 are drawn from the first set of simulations which used all available intron and exon data. The portions of the fractal plotted differ substantially for introns and exons. Interestingly this chaos automata, generated to separate introns and exons, separates the two classes of random data used in earlier simulations extraordinarily well, better than many of the fractals generated to separate the random sources. Fig. 7 shows several of the chaos automata fractals trained on the available data less the reserved cross-validation set, and their behavior on that cross-validation set. The fractals visually separate the

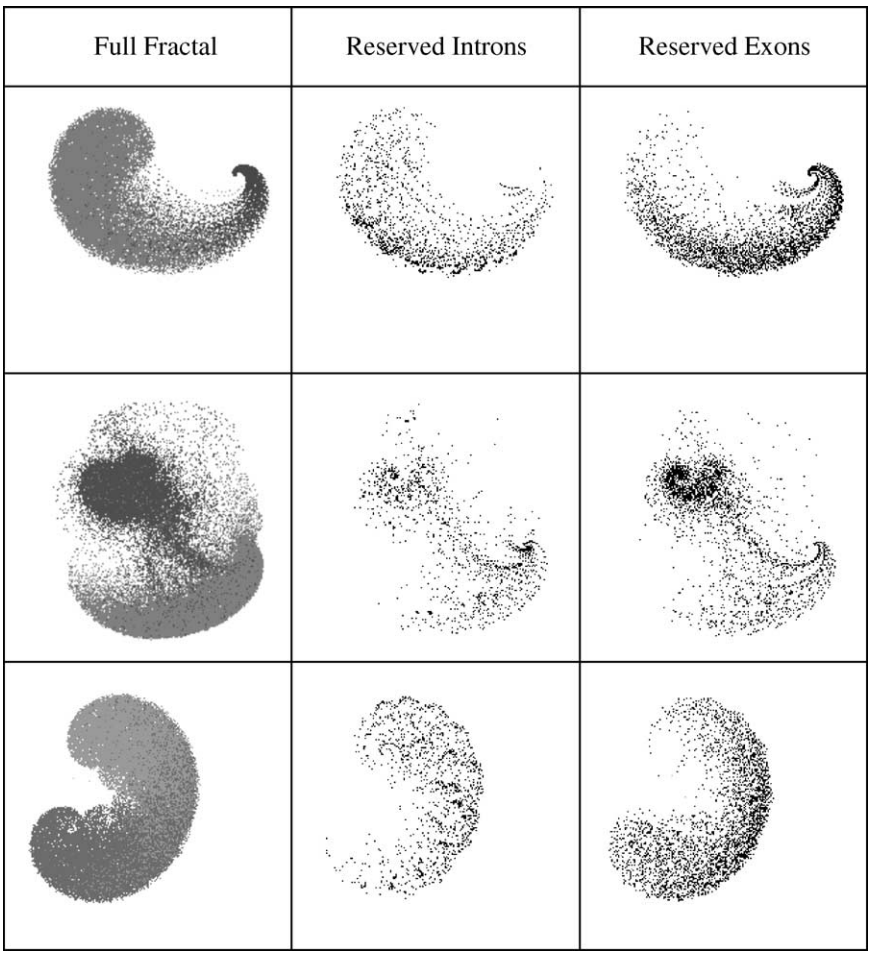

Fig. 7. The full fractal run on the training data, the intron cross-validation data, and the exon cross-validation data.

intron and exon data in the cross-validation data set. This separation is visible even in the black-and-white images displayed here. When the image is colored, the visual separation is more pronounced.

6. Conclusions

The work presented here demonstrates that chaos automata "programmed" (with parameters chosen) by an evolutionary algorithms can separate distinct classes of data. Since one chaos automata contains 33 integer and 32 real parameters this is a fairly substantial optimization task. The cross-validation experiments show that the evolutionary algorithm/chaos automata system managed to generalize and visually display some features that distinguish exons and introns. It is apparent to the authors that there is a qualitative increase in the degree of visual separation achieved with the chaos automata system over the earlier lookup-table based system described in [1]. This qualitative sense that there is an improvement is in agreement with theory. Lookup tables do not permit the kind of internal memory available to a system that includes state information. One feature of this research is that we begin to survey the possible fitness functions yielding fractals that "look good". Looking good is a qualitative state and so difficult to document. Interested parties that wish to examine the fractals on which the judgments in the paper are based may view all the fractals generated, in color, as well as preliminary result for continuing research, at: http://www.math.iastate.edu/danwell/Caut.html.

Given the qualitative nature of the goal, visual separation of data classes, there was relatively little mathematical theory to guide the design of the fitness functions. In the absence of a theory, intuition tempered by trial and error experimentation was sufficient to achieve satisfactory results. Having said this, it would be nice to try several other fitness functions and see what sort of results eventuate.

From comparing final fitness values to the authors' perception of the quality of visual separation of the two data types in question it is clear that the visual separation is not directly addressed by any of the fitness functions. There exist high fitness chaos automata that generate fractals that spatially mix the data plotted for each data type. We deduce that our fitness functions permit visual separation to arise but do not require it.

Examining the fitness values of the end-of-simulation chromosomes with the highest fitness as well as the fractals that result from the simulation reveal something of the character of the fitness landscape associated with each of the fitness functions. For fitness function F_4, in the simulations attempting to separate two random sources, the fitness of the best chaos automata in each simulation varied from 2.30 to 379, spanning two orders of magnitude. The appearances of the 50 fractals produced contained a large number of different morphological types. Both these observations suggest that the fitness landscape is very rugged, with many local optimum surrounded by high barriers relative to the variation operators chosen. When F_4 was used in the simulations driven by intron and exon data the variability in best-of-simulation fitness remained, with a minimum of 0.808 and a maximum of 578. The morphological diversity of the resulting fractals was also sustained.

One source of visual separation not visible in this paper is chromatic separation. Moving points associated with a given data type were given a particular color. This enabled the researchers to see the degree of separation of different data types when both were being plotted. A natural question to ask is whether the chromatic component can be sustained when the data type is not known. The answer seems to be yes. In each simulation, in addition to a separating fractal displaying the behavior of the chaos automata on the two data types, a *trace* fractal, colored in a manner that displays the chaos automata's internal state for each plotted point, was produced. These trace fractals appear on the web site given previously. These trace fractals indicate that some of the chaos automata separate the data types at the state level and so permit putative data type identification. Hence, color derived from state information could be used to give a tentative data type while visualizing DNA.

While some of the chaos automata associated data type with particular states others achieved separation of data types without and implicit assignment of data

type to state. Some states were involved in a nontrivial number of points of both data types. Since some of the automata did associate data type with state, such association could be incorporated into new fitness functions. The states used in plotting points of each data type would be tracked and, if the sets of states associated with each type of data were disjoint (or numerically close to disjoint) a fitness reward would result.

The technique presented here for training chaos automata showed a surprising generalization from *Zea mays* to synthetic data, shown in Fig. 5. In general, though, there is no reason to think that chaos automata trained on data from one organism will give informative visualizations on another organism.

6.1. Potential applications

The chaos game yields plots derived from DNA data with a clear meaning based on nucleotide, dinucleotide and trinucleotide frequencies [9]. Chaos automata, on the other hand, produce ad hoc visualizations of DNA data that can be selected (by choosing the correct training data) to separate particular sequence features of interest. The visualization created by chaos automata do not display *only* the features selected as interesting by the choice of training data, however. This suggests then that one potential application is in sequence viewer software. A small window would be added to the viewer that displays one or more fractal visualizations generated by chaos automata trained on features of interest within the data set being viewed. These fractals would then provide visual cues derived from the data. Their meaning may well be obscure, but these cues may help the users of the sequence viewer to classify and group sequences in new ways. This application is speculative.

7. Future work

Chaos automata are iterated function systems with finite state memory that generate interesting pictures. A good future goal is to learn how to clarify the meaning of these pictures. A human factors study on sequence type identification tasks done with and without the visualization described in Section 6.1 would be a nice direction for this research.

An obvious direction for future work are to challenge the chaos automata/evolutionary algorithm system with more types of data and continue the search for fitness functions that are more reliable in their production of visual separation. In line with this second goal, a study of the utility of chaos automata fractals in helping people to separate classes of data is an early priority. At the present time the phrase “visual separation of data” is intuitive, including spatial separation of points plotted for different data types and shape difference in the subset of the fractal associated with each data type. A more rigorous definition is desirable and is likely to lead to superior fitness functions.

As chaos automata incorporate finite state machines, a number of clever techniques for manipulating evolving finite state machines could be incorporated into the effort. The decomposition into sub-machines as defined in [5] is a good candidate for modularizing chaos automata with sub-automata that recognize various sorts of data. Such modularization would ease synthesis via hybridization of automata evolved on different data types. Another approach that may bear fruit is to attempt a cellular encoding of the chaos automata. *Cellular encoding* is a technique in which a complex structure is encoded by a linear set of instructions for constructing structures, instead of in the direct fashion specified in this paper.

While chaos automata have been tested on biological data in initial studies their ability to generate fractal fingerprints may be useful in a broad variety of domains. The problem of telling if two things are similar or different is one of the fundamental issues in information science. The techniques for solving these problems range from expert examination to automatically trained artificial neural nets. Chaos automata generalize the visual techniques built around iterated function systems.

References

[1] D. Ashlock, J.B. Golden, Iterated function system fractals for the detection and display of DNA reading frame, in: Proceedings of the 2000 Congress on Evolutionary Computation, 2000, pp. 1160–1167.

[2] T. Back, U. Hammel, H.-P. Schwefel, Evolutionary computation: comments on the history and current state, IEEE Trans. Evol. Comput. 1 (1) (1997) 3–17.
[3] P. Baldi, S. Brunak, Bioinformatics, The Machine Learning Approach, MIT Press, Cambridge, MA, 1998.
[4] M.F. Barnsley, Fractals Everywhere, Academic Press, Cambridge, MA, 1993.
[5] K. Chellapilla, D. Czarnecki, A preliminary investigation into evolving modular finite state machines, in: Proceedings of the 1999 Congress on Evolutionary Computation, 1999, pp. 1349–1356.
[6] C. Dutta, J. Das, Mathematical characterization of chaos game representations, J. Mol. Biol. 228 (1992) 715–719.
[7] K. Falconer, Techniques in Fractal Geometry, Wiley, New York, NY, 1997.
[8] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA, 1989.
[9] N. Goldman, Nucleotide, dinucleotide and trinucleotide frequencies explain patterns observed in chaos game representations of DNA sequences, Nucl. Acid Res. 21 (10) (1993) 2487–2491.
[10] J.H. Holland, Adaption in Natural and Artificial Systems, MIT Press, Cambridge, MA, 1992.
[11] H. Joel Jeffrey, Chaos game representation of gene structure, Nucl. Acid Res. 18 (8) (1990) 2163–2170.
[12] J.R. Koza, Genetic Programming, MIT Press, Cambridge, MA, 1992.
[13] J.R. Koza, Genetic Programming II, MIT Press, Cambridge, MA, 1994.
[14] J.L. Oliver, P. Bernaola-Galvan, J. Guerrero-Garcia, R. Romal Roldan, Entropic profiles of DNA sequences through chaos-game-derived images, J. Theor. Biol. 30 (1993) 457–470.
[15] V.W. Porto, N. Saravanan, D. Waagen, A.E. Eiben, Evolutionary Programming VII, Springer, New York, 1999.
[16] J. Setubal, J. Meidanis, Introduction to Computational Molecular Biology, PWS Publishing, Boston, MA, 1997.
[17] V.V. Solovyev, Fractal graphical representation and analysis of DNA and protein sequences, Biosystems 30 (1993) 137–160.
[18] G. Syswerda, A study of reproduction in generational and steady state genetic algorithms, in: Foundations of Genetic Algorithms, Morgan Kaufmann, Los Altos, CA, 1991, pp. 94–101.