Genetic algorithms in data analysis
Lankhorst, Marc Martijn

IMPORTANT NOTE: You are advised to consult the publisher’s version (publisher’s PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
1996

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.
Iterated Function Systems
Optimization

SOMEWHA T LOOSELY, one could define fractals as geometric figures that exhibit self-similarity at all levels of detail. Iterated function systems [11] are a way of generating fractals by means of a set of contractive transformations (see Section 7.1). This chapter investigates whether genetic algorithms can be useful in the construction of iterated function systems that approximate a given data set. The task of constructing an iterated function system that corresponds to a given image is referred to as the inverse problem. This problem is attacked by using a genetic algorithm to find an iterated function system’s coefficients.

The chapter starts with an introduction to iterated function systems, Section 7.1, and describes the approximation of one-dimensional and two-dimensional data using these systems. Next, Section 7.2 presents related work in combining genetic algorithms and iterated function systems. Furthermore, Section 7.3 discusses the adaptations of the standard genetic algorithm to this application, i.e., the representation of iterated function systems in the GA, the operators used on this representation, and the fitness function used. Finally, Section 7.4 gives the experiments that were conducted on both 1D and 2D input data, and Section 7.5 contains the conclusions that can be drawn from the results of these experiments.

This chapter was published as [103].

7.1 Iterated Function Systems

Iterated function systems (IFSs), developed by Hutchinson, Barnsley and others [79, 11, 10, 14], form a way of constructing fractals by means of a set of transformations of an image onto itself. These transformations are so-called contraction mappings. A transform \( f : X \rightarrow X \) on a space \( X \) with distance metric \( d \) is called a contraction mapping if there is a constant \( 0 \leq s < 1 \) (called a contractivity factor) such that

\[
d(f(x), f(y)) \leq s \cdot d(x, y) \quad \forall x, y \in X .
\]  

(7.1)

Intuitively, if a transformation \( f \) is contractive, the area of the image of the transformation is smaller than the area of the original.

Let \( (X, d) \) be a metric space and let \( (\mathcal{H}(X), h) \) denote the corresponding space of non-empty compact subsets, with the Hausdorff metric \( h \) as defined in [10]. A fractal can be defined as the fixed point of a contractive transformation on this space [11, p. 80].
Let \( w_i : X \to X \) \((i = 1, 2, \ldots, n)\) be contraction mappings on \((X, d)\) with contractivity factors \( s_i \), and define \( w_i : \mathcal{H}(X) \to \mathcal{H}(X) \) by

\[
    w_i(A) = \{ w_i(p) \mid p \in A \} \quad \text{for} \ A \in \mathcal{H}(X) .
\]

Then \( W : \mathcal{H}(X) \to \mathcal{H}(X) \), defined by

\[
    W(A) = \bigcup_{i=1}^{n} w_i(A) ,
\]

is a contraction mapping with contractivity factor \( s = \max \{ s_i \mid i = 1, 2, \ldots, n \} \), and the corresponding IFS is denoted \( \{ X; w_i, i = 1, 2, \ldots, n \} \).

The transformations commonly used in IFSs in the Euclidean plane are affine, i.e., they are of the form

\[
    w_i(p) = w_i \left( \begin{array}{c} x \\ y \end{array} \right) = \left( \begin{array}{cc} a_i & b_i \\ c_i & d_i \end{array} \right) \left( \begin{array}{c} x \\ y \end{array} \right) + \left( \begin{array}{c} e_i \\ f_i \end{array} \right) = A_i \cdot p + t_i . \tag{7.4}
\]

Repeatedly applying the IFS \( W \) as defined in (7.3) to a finite image \( A \), we approximate the fixed point\(^1\) \( \mathcal{A} \) of \( W \) given by

\[
    \mathcal{A} = \lim_{n \to \infty} W^n(A) ,
\]

which is called the attractor of \( W \). This attractor is independent of the image \( A \) to which \( W \) is applied. This means that the attractor is completely defined by the coefficients of \( W \), and that it can be reproduced by repeated application of the IFS to any starting point.

If the transformations of an IFS \( W \) are continuous in some parameter \( r \), the attractor of \( W \) will also be continuous in \( r \) [10, p. 113]. This is very important, because it tells us that we can continuously control the attractor of an IFS by adjusting parameters in the transformations. It also means that we can smoothly interpolate between attractors.

The Collage Theorem proved in [12] states that any image can be represented by a collage of smaller copies of that image that is arbitrarily close to the original image, i.e., for any given image \( A \) there exists an IFS \( W \) with attractor \( \mathcal{A} \) such that \( \mathcal{A} \) is arbitrarily close to \( A \). The problem of finding an IFS that defines this collage for an arbitrary image is often referred to as the inverse problem. No efficient methods to solve the inverse problem are known. In this chapter, we apply a genetic algorithm to address this problem.

### 7.1.1 IFS Approximation of 1D Signals

If we want to approximate a 1D signal \( D = \{(x_j, F_j) \in \mathbb{R}^2 \mid x_0 < x_1 < \cdots < x_N, \ j = 0, 1, \ldots, N\} \), using an IFS with affine contraction mappings as defined in Section 7.1, we need to restrict the affine transformations in such a way that the resulting attractor \( \mathcal{A} \subset \{(x, y) \in \mathbb{R}^2 \} \) is the graph of a continuous function \( f : [x_0, x_N] \to \mathbb{R} \). To ensure this, we use an IFS of the form \( \{ \mathbb{R}^2; w_i, i = 1, 2, \ldots, n \} \) where the contraction mappings \( w_i \) are shear transformations: they have the special structure

\[
    w_i \left( \begin{array}{c} x \\ y \end{array} \right) = \left( \begin{array}{cc} a_i & 0 \\ c_i & d_i \end{array} \right) \left( \begin{array}{c} x \\ y \end{array} \right) + \left( \begin{array}{c} e_i \\ f_i \end{array} \right) . \tag{7.6}
\]

\(^1\)The existence and uniqueness of this fixed point is guaranteed by Banach’s Fixed Point Theorem [59]

\(^2\)Each of the \( w_i \)’s has itself a fixed point: if we write \( w_i(p) = A_i \cdot (p - q) + q \) then \( q \) is the fixed point of \( w_i \).
These shear transformations map lines parallel to the y-axis into lines parallel to the y-axis. Furthermore, these transformations should be just-touching: we want to have a set of points $S = \{(p_i, q_i) \in \mathbb{R}^2 \mid p_0 < p_1 < \cdots < p_n, \ i = 0, 1, \ldots, n\}$, such that the transformations touch one another at each end point:

$$w_i \left( \begin{array}{c} p_0 \\ q_0 \end{array} \right) = \left( \begin{array}{c} p_{i-1} \\ q_{i-1} \end{array} \right),$$

$$w_i \left( \begin{array}{c} p_n \\ q_n \end{array} \right) = \left( \begin{array}{c} p_i \\ q_i \end{array} \right).$$

This implies that each transformation $w_i$ must obey the following four linear equations:

$$a_i p_0 + e_i = p_{i-1},$$

$$a_i p_n + e_i = p_i,$$

$$c_i p_0 + d_i q_0 + f_i = q_{i-1},$$

$$c_i p_n + d_i q_n + f_i = q_i.$$ 

It follows that there is effectively one free parameter in each transformation. We take this parameter to be $d_i$ because this allows us to specify the vertical scaling of the shear transformation: The ratio of the length of a vertical line segment $L$ to the length of its image $w_i(L)$ is $|d_i|$. The other parameters are computed from $d_i$ and $S$. Using $\Delta = p_n - p_0$, we have:

$$a_i = (p_i - p_{i-1})/\Delta,$$

$$c_i = (q_i - q_{i-1})/\Delta - d_i(q_n - q_0)/\Delta,$$

$$e_i = (p_n p_{i-1} - p_0 p_i)/\Delta,$$

$$f_i = (p_n q_{i-1} - p_0 q_i)/\Delta - d_i(p_n q_0 - p_0 q_n)/\Delta.$$ 

From (7.10) it follows that $0 < a_i < 1$, and to ensure contractivity it suffices to choose $d_i$ such that $|d_i| < 1$.

Barnsley [10] uses shear transformations to interpolate a given set of data: he chooses $S = D$. If we have a large data set, this approach seems less appropriate, because we end up with a large number of transforms that do not tell us much about the global structure of the data. We would rather like to find a set of transformations and a set $S$ that is much smaller than $D$, such that the attractor of the corresponding IFS approximates the data set. However, this is not a trivial task: we have to optimize both the set of points $S$, and the parameters $d_i$.

### 7.1.2 IFS Approximation of 2D Images

To approximate 2D images using an IFS as defined in (7.3), we have to restrict its parameters to satisfy the contractivity condition (7.1). This amounts to ensuring that

$$\text{(area of } w_i(B)\text{)} < \text{(area of } B\text{)}.$$

Because

$$\text{(area of } w_i(B)\text{)} = |\det A_i| \cdot \text{(area of } B\text{)}.$$

This is guaranteed if we choose $d_i$ such that $|d_i| < 1$. Therefore, the choice of point-set $S$ and parameter $d_i$ is determined uniquely by the data set $D$. Furthermore, since $D$ contains $n + 1$ points, $S$ must contain at most $n$ points.
(with \( A_i \) as in (7.4)), we have
\[
|\det A_i| < 1, \tag{7.13}
\]
which can be fulfilled by
\[
-\frac{1}{2} \sqrt{2} \leq a_i, b_i, c_i, d_i \leq \frac{1}{2} \sqrt{2}. \tag{7.14}
\]
Since we want to map an image \( A \in [x_{\text{min}}, x_{\text{max}}] \times [y_{\text{min}}, y_{\text{max}}] \) onto itself, the translation parameters \( e_i \) and \( f_i \) must be restricted by the borders of the image:
\[
x_{\text{min}} \leq e_i \leq x_{\text{max}},
\]
\[
y_{\text{min}} \leq f_i \leq y_{\text{max}}. \tag{7.15}
\]
Note that these parameter bounds do not prevent that points of the image are transformed to points outside the region \([x_{\text{min}}, x_{\text{max}}] \times [y_{\text{min}}, y_{\text{max}}]\). However, a well-chosen fitness function will quickly eliminate such spurious transformations.

## 7.2 Related Work

Until now, no efficient algorithm to solve the inverse problem is known. Many approaches to the problem have been tried, some of which are based on evolutionary algorithms.

In [60], Giles describes both a deterministic and a genetic approach to IFS construction. The work stresses the importance of the GA’s fitness function characteristics, but its implementation suffers from the prohibitively long running time resulting from the use of the Hausdorff metric as given in [10] in the algorithm’s fitness evaluation.

Hoskins and Vagner [76] apply evolutionary programming [52, 50] to find IFSs of images, aiming at image compression. Their algorithm is supplied with a set of contraction mappings (i.e., the matrices \( A_i \) of (7.4) below), but has to evolve the fixed point of each mapping itself. Shonkwiler et al. [140] presents a genetic algorithm for solving the 1D inverse problem for very small problems (typically 6 parameters).

An analysis of error bounds for IFS optimization is used by Levy-Vehel and Lutton [109] to estimate the accuracy needed in representing IFSs in a GA. The resulting algorithm seems moderately successful on 2D approximations, but no performance statistics are given.

Vences and Rudomin [150] apply GAs to the related problem of image compression using IFSs [14, 13]. This does not solve the inverse problem, but a related one which partitions an image into small blocks (so-called domain blocks) and tries to find optimal mappings between these domain blocks and larger so-called range blocks. They use GAs for compression of both single images and image sequences.

The main differences from the above of the approach of this chapter to the inverse problem are found in the fitness functions and encodings used, and in that both 1D signals and 2D binary images are approximated. The main goal of this research is to investigate whether a GA can produce approximations of sufficient quality in a reasonable amount of time.

## 7.3 A Genetic Algorithm for Fractal Approximation

### 7.3.1 Representation

To be able to use a genetic algorithm to find optimal parameters for IFS approximations, we need to represent the transformation parameters mentioned in Sections 7.1.1 and 7.1.2 as chro-
mosomes. Each chromosome will represent one parameter set, and the real-valued parameters form the genes.

On the lowest level, these real numbers are represented as bitstrings; a simple, linear encoding suffices if an upper and lower limit of the real numbers are known. Each gene has three parameters that determine its mapping into a real number:
- upper limit \( p \);
- lower limit \( q \);
- number of bits \( \ell \).

Using these parameters, every coefficient \( w \) is encoded as:

\[
e(w) = \lfloor (2^\ell - 1) \cdot (w - p)/(q - p) \rfloor.
\]

In all experiments, the real-valued parameters are encoded in 16 bits.

**Signal Approximation**

In the case of IFS approximation of signals (Section 7.1.1), we have to encode the set of points \( S \) and the parameters \( d_i \) of the shear transformations as defined in (7.7) and (7.8). If \( n \) transformations are used, \( S \) contains \( n + 1 \) points. However, we know that \( p_0 = x_0 \) and \( p_n = x_n \), since the domain \([p_0, p_n]\) of the approximation should be the same as the domain \([x_0, x_n]\) of the signal. Furthermore, the definition of \( S \) states that \( p_0 < p_1 < \cdots < p_n \). We could encode absolute positions for the \( p_i \)'s, but that would entail sorting the parameters in the decoding process. Moreover, a mutation of some \( p_i \) may cause this end point to trade places with the end point of one of the neighboring transformations, resulting in a major change in the decoded IFS. To avoid these problems, the position of each \( p_i \) is encoded by giving a displacement \( \Delta_i \) relative to its predecessor:

\[
\begin{align*}
p_0 &= x_0, \\
p_n &= x_n, \\
p_i &= p_{i-1} + (p_n - p_0)\Delta_i/\sum_{j=1}^{n} \Delta_j, \quad \Delta_i > 0, \quad i = 1, \ldots, n.
\end{align*}
\]

The chromosomes thus consist of \( 3n + 1 \) real-valued genes: \( \{q_0, d_1, \Delta_1, q_1, \ldots, d_n, \Delta_n, q_n\} \). The \( q_i \)'s are limited by the range of the signal that is approximated; the range of the \( d_i \)'s is chosen to be \((-1, 1)\) since we want the transformations to be contractive (see Section 7.1.1).

**Image Approximation**

For image approximation, the six parameters of each affine transformation (7.4) must be encoded, using the ranges as derived in Section 7.1.2:

\[
\begin{align*}
-\frac{1}{2}\sqrt{2} \leq a_i, b_i, c_i, d_i \leq \frac{1}{2}\sqrt{2}, \\
x_{\text{min}} \leq e_i \leq x_{\text{max}}, \\
y_{\text{min}} \leq f_i \leq y_{\text{max}}.
\end{align*}
\]

Thus, an IFS consisting of \( n \) transformations is encoded in a chromosome of \( 6n \) real-valued genes: \( \{a_1, b_1, \ldots, e_n, f_n\} \).
7.3.2 Genetic Operators

Selection

Selection is carried out by a stochastic universal sampling algorithm [9]. Furthermore, a linear ranking selection scheme [153] is used in combination with an elitist strategy in which the best (w.r.t. the fitness criterion given in Section 7.3.3) individual of the population always survives to the next generation.

Mutation

Three types of real-number mutation are used:

1. Random mutation: a real number \( r \in [p, q] \) is mapped to another real number \( r' \in [p, q] \) using a uniform random distribution;
2. Gaussian big creep: a real number \( r \in [p, q] \) is mapped to \( r' = r + N(0, \sigma_{\text{big}}) \) (and clipped to \( [p, q] \)) where \( \sigma_{\text{big}} = 0.2 \cdot (q - p) \);
3. Gaussian little creep: a real number \( r \in [p, q] \) is mapped to \( r' = r + N(0, \sigma_{\text{little}}) \) (and clipped to \( [p, q] \)) where \( \sigma_{\text{little}} = 0.01 \cdot (q - p) \).

The latter two are variants of Davis’ big creep and little creep operators [39] (see also Section 2.4.3). Each of these types has its own merits. Random mutation is useful to maintain a certain level of genetic diversity in the population to prevent premature convergence, big creep makes jumps in a fairly large neighborhood of a point in the search space, and little creep serves to optimize already good values by making small changes.

The adaptive scheme of Section 6.4.3 (see Figure 6.9) is used to determine the frequency ratio of the different mutation types. The total mutation probability remains constant, but the relative application frequencies of the three types change over time. In Chapter 6 this scheme was shown to outperform a scheme with a fixed proportion of the different mutation types.

In the experiments comparing bit to real-number operators, standard binary mutation (see Section 2.2.3) is also used.

Recombination

On the binary representation, standard two-point crossover as defined in Section 2.4.2 is used.

For real numbers, arithmetical crossover is employed. Arithmetical crossover as defined in [115] takes two (real-valued) parent genes \( s \) and \( t \) and computes their offspring genes \( s' \) and \( t' \) as a linear combination of the parents’ values:

\[
\begin{align*}
s' &= a \cdot s + (1-a) \cdot t \\
t' &= (1-a) \cdot s + a \cdot t 
\end{align*}
\]

using a parameter \( a \in [0, 1] \).

In our case, for each individual gene participating in the crossover, the parameter \( a \) is a uniformly random choice from the interval \([0, 1]\). We use a combination of two-point and arithmetical crossover: two crossover points are chosen randomly, and the arithmetical crossover takes place between these two points.
7.3.3 Fitness Evaluation

Signal approximation

To evaluate the quality of the individuals in the population, we need a fitness function. In the case of signal approximation, the natural choice is to use some norm of the difference between the signal \( f \) and the approximation \( \hat{f} \) as encoded on the chromosome; therefore, we use a fitness function based on the \( L^2 \) norm of the error, divided by the \( L^2 \) norm of the signal:

\[
F_f(\hat{f}) = \frac{\|f - \hat{f}\|_2}{\|f\|_2}.
\]

Image approximation

For binary image approximation, we can compute the fitness based on the (relative) number of point matches between the image \( A \) and either the collage \( W(A) \) or the attractor \( \mathcal{A} \). To compute the relative number of matches, we have to look at the mismatches first.

There are two types of mismatch: points of the image not covered by the attractor, and points of the attractor that do not cover the image. To maximize the coverage between the image and the attractor, both these mismatches are to be minimized. However, both types can be weighted differently, giving the following fitness function that is to be maximized:

\[
F_A(B) = p_1 \cdot \left(1 - \frac{|A \cap B|}{|A|}\right) + p_2 \cdot \left(1 - \frac{|\mathcal{A} \cap B|}{|B|}\right),
\]

where \( B \) equals either the collage \( W(A) \) or the attractor \( \mathcal{A} \), and \( p_1 \) and \( p_2 \) are the weight factors for the two types of mismatch.

It seems preferable for the attractor to quickly cover a large part of the image. Any attractor points outside the image may be eliminated later on in the optimization process. Such behavior might be achieved by choosing \( p_1 > p_2 \): correctly covering a point of the image leads to a larger gain than correctly missing a point outside the image, and only when after some generations a sufficient part of the image is covered, it becomes profitable to eliminate spurious points. Based on these considerations and after some preliminary experiments, the parameters were set to \( p_1 = 1 \) and \( p_2 = 0.2 \).

7.4 Experiments

7.4.1 Test Data

To test the performance of the GA in finding IFS approximations of data, we used several test sets. The first test set, a one-dimensional signal constructed from \( B_2 \)-splines, is artificial but does not contain deliberate, built-in self-similarities\(^3\). This signal was used to evaluate whether this type of approximation is in general suitable for signal approximation. It is shown in Figure 7.1, and will be approximated using 10 segments.

\(^3\)The Collage Theorem (Section 7.1) states that any signal can be built up from smaller copies of itself. In this case, however, the signal is not constructed in this way.
The second and third sets are artificial signals with built-in self-similarity which were used to assess whether self-similarities are indeed found by the GA. Signal 2, depicted in Figure 7.2, has 5 self-similar segments, delimited by the vertical lines in the figure. Signal 3, depicted in Figure 7.3, has 10 self-similar segments. The fourth test set consists of real-life human glucose level data, and is shown in Figure 7.4. All signals are scaled to the interval [0, 1].

For the case of image approximation, two simple pictures were used, shown in Figures 7.5 and 7.6. The first is a 50 × 50 image of a triangle, that can be composed of three transforms of itself. The second is a 50 × 50 L-shaped image. It is left as an exercise to the reader to show that it can be composed of four transforms of itself.

**Figure 7.1:** Signal 1. Artificial, constructed from $B_2$-splines. 256 samples.

**Figure 7.2:** Signal 2. Artificial, with 5 self-similar segments. 256 samples.
7.4 Experiments

Figure 7.3: Signal 3. Artificial, with 10 self-similar segments. 256 samples.

Figure 7.4: Signal 4. Real-life glucose level data. 700 samples.

Figure 7.5: Image 1. Triangle. 50 × 50 pixels

Figure 7.6: Image 2. L-shape. 50 × 50 pixels
7.4.2 Experiments Regarding Parameters

To tune some of the parameters of the GA, it was tested on signal 1 using different mutation rates and population sizes. Since it is generally assumed (and theoretically supported, see Section 2.3) that GAs derive their search power from recombination of schemata by the crossover operator, the crossover probability is kept constant at a rate of 0.9.

Mutation Rate

For a population of 100 individuals, using binary and real-number operators, and limiting the number of function evaluations to 100,000, the results shown in Table 7.1 were obtained. This table contains the fitness (the relative $L^2$ norm of the error) of the best individual found after 100,000 evaluations, averaged over 50 runs of the algorithm. In this and following tables, a standard $t$-test is used to obtain the 95% confidence interval, from which the error margins are derived. A graph of these results is given in Figure 7.7.

<table>
<thead>
<tr>
<th>mut.rate</th>
<th>binary</th>
<th>real-number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0001</td>
<td>$8.84 \cdot 10^{-2} \pm 2.1 \cdot 10^{-3}$</td>
<td>$1.25 \cdot 10^{-1} \pm 3.3 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>0.0002</td>
<td>$8.69 \cdot 10^{-2} \pm 2.2 \cdot 10^{-3}$</td>
<td>$1.10 \cdot 10^{-1} \pm 2.8 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>0.0005</td>
<td>$8.54 \cdot 10^{-2} \pm 2.5 \cdot 10^{-3}$</td>
<td>$9.64 \cdot 10^{-2} \pm 2.7 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>0.001</td>
<td>$8.27 \cdot 10^{-2} \pm 2.5 \cdot 10^{-3}$</td>
<td>$9.22 \cdot 10^{-2} \pm 2.5 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>0.002</td>
<td>$8.14 \cdot 10^{-2} \pm 1.9 \cdot 10^{-3}$</td>
<td>$8.82 \cdot 10^{-2} \pm 2.1 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>0.005</td>
<td>$8.56 \cdot 10^{-2} \pm 2.5 \cdot 10^{-3}$</td>
<td>$8.29 \cdot 10^{-2} \pm 1.8 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>0.01</td>
<td>$9.48 \cdot 10^{-2} \pm 2.3 \cdot 10^{-3}$</td>
<td>$8.12 \cdot 10^{-2} \pm 2.1 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>0.02</td>
<td>$1.05 \cdot 10^{-1} \pm 2.7 \cdot 10^{-3}$</td>
<td>$8.12 \cdot 10^{-2} \pm 2.4 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>0.05</td>
<td>$1.38 \cdot 10^{-1} \pm 3.2 \cdot 10^{-3}$</td>
<td>$7.69 \cdot 10^{-2} \pm 1.5 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>0.1</td>
<td>$1.62 \cdot 10^{-1} \pm 2.4 \cdot 10^{-3}$</td>
<td>$7.81 \cdot 10^{-2} \pm 1.9 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>0.2</td>
<td>$1.82 \cdot 10^{-1} \pm 2.7 \cdot 10^{-3}$</td>
<td>$7.78 \cdot 10^{-2} \pm 1.7 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>0.5</td>
<td>$1.92 \cdot 10^{-1} \pm 2.4 \cdot 10^{-3}$</td>
<td>$7.79 \cdot 10^{-2} \pm 2.1 \cdot 10^{-3}$</td>
</tr>
</tbody>
</table>

Table 7.1: Fitness of the best individual after 100,000 function evaluations, for approximation of signal 1, using binary and real-number operators, and a population size of 100. Results averaged over 50 runs.

From these results it can be concluded that for a population of 100 individuals, the GA performs best with a mutation rate around 0.002 if binary operators are used, and a rate around 0.05 for real-number operators. This first rate is in accordance [116], in which theoretical support for a mutation rate of $1/\ell$ on binary strings is given. If we apply this heuristic to the real-number case (although this is not theoretically justified), we arrive at a rate of 0.03, which is close to the empirically determined value of 0.05. Figure 7.8 depicts the algorithm’s behavior for these parameter settings.
7.4 Experiments

Figure 7.7: Fitness of best individual vs. mutation rate for approximation of signal 1, using binary and real-number operators. Population size 100, 100,000 function evaluations. Results averaged over 50 runs. Vertical lines denote 95% confidence interval.

Figure 7.8: Fitness of best individual vs. generations for approximation of signal 1, using binary and real-number operators, and mutation rates of 0.002 and 0.05, respectively. Population size 100, 100,000 function evaluations. Results averaged over 50 runs.

Population Size

To assess the dependency of the algorithm’s performance on the population size, it was tested on different populations using a fixed mutation rate. For the GA with binary operators, a mutation rate of 0.002 was used, and for the algorithm with real-number operators, a rate of
0.05 was used. Both these figures are obtained from the results given in the previous section.

The results obtained are given in Table 7.2, which contains the fitness (the relative $L^2$ error) of the best individual found after 100,000 function evaluations, averaged over 50 runs of the algorithm. Figure 7.9 shows a graph of these results. In both the binary and the real-number case, a population size of 50 seems to yield the best signal approximations.

<table>
<thead>
<tr>
<th>pop.size</th>
<th>binary</th>
<th></th>
<th>real-number</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>$8.17 \cdot 10^{-2}$ $\pm$ $2.3 \cdot 10^{-3}$</td>
<td>$7.64 \cdot 10^{-2}$ $\pm$ $2.0 \cdot 10^{-3}$</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>$8.03 \cdot 10^{-2}$ $\pm$ $2.2 \cdot 10^{-3}$</td>
<td>$7.58 \cdot 10^{-2}$ $\pm$ $1.9 \cdot 10^{-3}$</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>$8.14 \cdot 10^{-2}$ $\pm$ $1.9 \cdot 10^{-3}$</td>
<td>$7.69 \cdot 10^{-2}$ $\pm$ $1.5 \cdot 10^{-3}$</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>$8.84 \cdot 10^{-2}$ $\pm$ $1.9 \cdot 10^{-3}$</td>
<td>$8.49 \cdot 10^{-2}$ $\pm$ $1.9 \cdot 10^{-3}$</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>$1.05 \cdot 10^{-1}$ $\pm$ $1.5 \cdot 10^{-3}$</td>
<td>$1.05 \cdot 10^{-1}$ $\pm$ $1.8 \cdot 10^{-3}$</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>$1.27 \cdot 10^{-1}$ $\pm$ $1.9 \cdot 10^{-3}$</td>
<td>$1.27 \cdot 10^{-1}$ $\pm$ $1.7 \cdot 10^{-3}$</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>$1.53 \cdot 10^{-1}$ $\pm$ $2.3 \cdot 10^{-3}$</td>
<td>$1.45 \cdot 10^{-1}$ $\pm$ $1.8 \cdot 10^{-3}$</td>
<td></td>
</tr>
</tbody>
</table>

**Table 7.2:** Fitness of the best individual after 100,000 function evaluations, for approximation of signal 1, using binary and real-number operators, and mutation rates of 0.002 and 0.05, respectively. Results averaged over 50 runs.

![Figure 7.9](image)

**Figure 7.9:** Average fitness of best individual vs. population size, using binary and real-number operators, and mutation rates of 0.002 and 0.05, respectively. Results over 50 runs.

From these results and those of the previous section, it can be concluded that for binary operators, a mutation rate of 0.002 together with a population size of 50 are good settings, and for real-number operators, a 0.05 mutation rate and the same population size are suitable. Naturally, it cannot be guaranteed that these values are optimal, since that would require a (computationally very expensive) full search of the algorithm’s parameter space.

In any case, from both Table 7.1 and Table 7.2 we may infer that the use of real-number operators yields slightly better approximation results: the 95% confidence intervals do not
Figure 7.10: Fitness of best individual vs. generations for approximation of signal 1, using binary and real-number operators, and mutation rates of 0.002 and 0.05, respectively. Population size 50, 100,000 function evaluations. Results averaged over 50 runs.

overlap. This is illustrated in Figure 7.10, which clearly exhibits a significant advantage of real-number over binary operators; both faster convergence and better approximation quality are achieved. Consequently, these operators were used in the rest of the experiments.

7.4.3 Results on 1D Signals

Table 7.3 shows the average approximation quality achieved by the GA on the four test signals described in Section 7.4.1.

<table>
<thead>
<tr>
<th>Signal</th>
<th>segments</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>$7.58 \cdot 10^{-2} \pm 1.9 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>$8.95 \cdot 10^{-2} \pm 8.8 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>$1.32 \cdot 10^{-1} \pm 8.9 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>$1.03 \cdot 10^{-1} \pm 7.4 \cdot 10^{-3}$</td>
</tr>
</tbody>
</table>

Table 7.3: Relative $L^2$ norms of the error signal. Results over 50 runs. Population size 50, mutation rate 0.05 for signal 1, 3, and 4, and 0.025 for signal 2.

To test the GA’s ability to detect self-similarities, signals 2 and 3 were used as given in Section 7.4.1, Figures 7.2 and 7.3. Figures 7.11 and 7.12 show the behavior of the algorithm for signals 2 and 3, respectively.

Additionally, the algorithm was tested on the real-life signal 4 (Figure 7.4). The behavior of the algorithm for this signal is depicted in Figure 7.13.

Although in some cases a nearly perfect approximation was achieved, the average approximation is reasonably accurate on some of the self-similar segments, but it is inaccurate on one or more other segments. A typical example is shown in Figure 7.14: the first two segments
of signal 2 are approximated very poorly, but the approximation of the other three segments is quite good. This effect is exhibited even more by the approximations of signal 3, which explains their poorer average quality.

Quite unexpectedly, the average approximation of signal 4 was better than that of signal 3, although the latter contains artificial self similarities, and the former consists of real-life data. Apparently the self-similar segments of signal 3 are too small for the algorithm to detect.
7.4 Experiments

Figure 7.13: Fitness of best individual vs. generations for approximation of signal 4. Mutation rate 0.05, population size 50, 100,000 function evaluations. Results averaged over 50 runs.

Figure 7.14: Approximation of signal 2. Vertical lines denote the boundaries of self-similar segments of the approximation.

7.4.4 Results on 2D Images

Table 7.4 shows the average approximation quality achieved by the GA on the two test images described in Section 7.4.1. The quality of the approximations of Figures 7.5 and 7.6, as exemplified by Figures 7.15 and 7.16, is quite poor. Convergence graphs are presented in Figures 7.17 and 7.18.
<table>
<thead>
<tr>
<th>Image</th>
<th>mappings</th>
<th>runs</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>50</td>
<td>$7.98 \cdot 10^{-1} \pm 7.4 \cdot 10^{-3}$</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>50</td>
<td>$7.90 \cdot 10^{-1} \pm 4.6 \cdot 10^{-3}$</td>
</tr>
</tbody>
</table>

Table 7.4: Point coverage (as defined in (7.21)) between image and attractor. Population size 50, mutation rate 0.05.

Figure 7.15: Approximation of image 1. 50×50 pixels.

Figure 7.16: Approximation of image 2. 50×50 pixels.

Figure 7.17: Fitness of best individual vs. generations for approximation of image 1. Mutation rate 0.05, population size 50, 100,000 function evaluations. Results averaged over 50 runs.
7.5 Discussion

The results presented in the previous section show that approximations of both signals and images found by the genetic algorithm leave much to be desired. Although the 1D signal approximations are in some cases reasonably accurate, the approximation quality of the 2D images is unsatisfactory.

A plausible reason for this behavior appears to be that the building block hypothesis does not hold. Since contraction mappings and their coefficients only have significance when combined in an IFS, determining the fitness contributions of isolated mappings or (sets of) coefficients is meaningless. Combining building blocks with above average fitness does not necessarily lead to a highly fit IFS; there is no relation between the building blocks’ fitness and the fitness of the IFS.

This implies that there are no ‘good’ or ‘bad’ building blocks; the fitness of a building block depends completely on its context, the surrounding IFS. The problem suffers from a severe form of epistasis: all parts of the genotype interact to form the phenotype, and one cannot identify parts of the genotype that encode for isolated phenotypic traits. It follows that crossover will not operate effectively on this problem.

In the 1D signal approximation case, both the segment end points (see Section 7.3.1) and a single IFS parameter \(d_i\) were encoded for each transformation. These end points are more or less independent of the IFS parameters, and therefore the epistasis is less severe in this case, and crossover operates more efficiently. This explains the better performance of the GA in the 1D case.

To verify the explanation given above the correlation between parent and offspring fitness was computed for all test sets and a population of 1000 random parents. The results are summarized in Table 7.5. As expected, correlations are significantly higher in the 1D case. For 2D approximations, the correlation coefficients’ low values of around 0.50 indicate that crossover
<table>
<thead>
<tr>
<th>Test case</th>
<th>correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal 1</td>
<td>0.68</td>
</tr>
<tr>
<td>Signal 2</td>
<td>0.68</td>
</tr>
<tr>
<td>Signal 3</td>
<td>0.65</td>
</tr>
<tr>
<td>Signal 4</td>
<td>0.74</td>
</tr>
<tr>
<td>Image 1</td>
<td>0.48</td>
</tr>
<tr>
<td>Image 2</td>
<td>0.55</td>
</tr>
</tbody>
</table>

**Table 7.5:** Fitness correlation coefficients between parents and offspring for crossover.

Figure 7.19: Fitness of best individual vs. generations for approximation of image 2, with and without using crossover. Mutation rate 0.05, population size 50, 100,000 function evaluations. Results averaged over 50 runs.

does not contribute significantly to the performance of the GA; it would do equally well without crossover. This has been verified by running tests, both with and without crossover, for approximation of Figure 7.6. As can be seen from the convergence graphs of Figure 7.19, the use of crossover indeed makes no significant difference: the 95% confidence intervals overlap.

From the above we may conclude that genetic algorithms are not particularly suited to the problem of finding an IFS to approximate a given 2D data set. Recombination of parts of two different IFSs by means of the crossover operator does, in most cases, not result in an IFS with a fitness close to its parents’ fitnesses. Evolutionary programming [52, 50], which does not employ crossover, might be more fit for this problem than genetic algorithms.