Fractals, Graphs, and Fields

Franklin Mendivil

1. INTRODUCTION. One of the most amazing facets of mathematics is the experience of starting with a problem in one area of mathematics and then following the trail through several other areas to the solution (or several versions of the solution). We illustrate this with a problem that starts out as a problem in rendering the attractor of an Iterated Function System (IFS), which leads to a solution that involves finding an Eulerian cycle in a certain graph and then to finding generators for the multiplicative group of a finite field.

We start with an introduction to IFS fractals and the problem of generating an image of the attractor of an IFS. For a more complete introduction to the theory of Iterated Function Systems, we refer to reference [**1**].

2. ITERATED FUNCTION SYSTEMS. The basic idea behind Iterated Function System fractals is that we wish to formalize the concept of *self-similarity*. That is, given an image like Figure 1, we wish to formalize our notion that the set is made up of three smaller copies of itself.

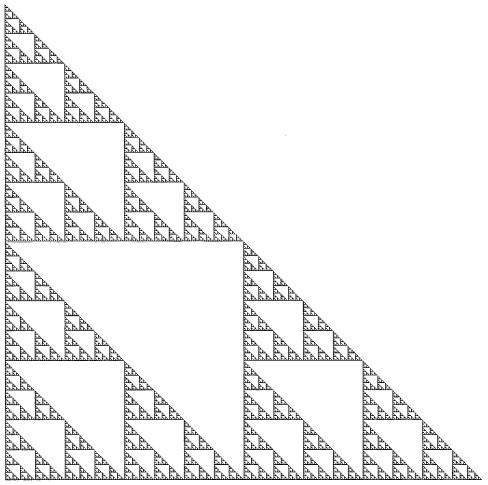

Figure 1. The Sierpiński gasket.

As a motivating example, consider the three maps from the unit square to itself defined by

$$w_0(x, y) = \left(\frac{x}{2}, \frac{y}{2}\right), \quad w_1(x, y) = \left(\frac{x+1}{2}, \frac{y}{2}\right), \quad w_2(x, y) = \left(\frac{x}{2}, \frac{y+1}{2}\right).$$

Clearly each of these maps contracts in all directions by a factor of two. Thus, the image of Figure 1 under w_0 is the smaller copy in the lower left corner. Similarly, the image under w_1 is the smaller copy in the lower right corner and under w_2 is the

smaller copy in the upper left corner. Thus, this collection of maps gives us some way to codify the fact that Figure 1 is made up of three smaller copies of itself. The set in Figure 1 is called the *Sierpiński gasket.*

Formally, an *Iterated Function System* is a collection $\mathcal{W} = \{w_i : X \to X, i = 0, 1, \ldots, N-1\}$ of a finite number N of contractive self-maps of a complete metric space (X, d). For the purposes of generating fractal images we usually take X to be the unit square in $\mathbb{R}^2$; that is,

$$X = [0, 1]^2 = \{(u, v) : 0 \leq u, v \leq 1\}$$

and d is the usual Euclidean distance. However, the theory applies equally well to any complete metric space.

Now a function $w : X \to X$ (returning to the general setting) is a *contraction* with *contraction factor* $s (0 \leq s < 1)$ provided two things are true: (1) $d(w(x), w(y)) \leq sd(x, y)$ for all x and y in X and (2) s is the smallest number for which (1) holds. For an IFS $\{w_i\}$, we will let s_i be the contraction factor of w_i and define $s = \max_i s_i$ to be the contraction factor for the IFS.

Appealing to our example of the Sierpiński gasket for motivation, we use the IFS $\mathcal{W} = \{w_i\}$ to define a set-map W on X that "combines" the action of all the individual maps w_i; namely, we define

$$W(B) = \bigcup_i w_i(B)$$

for each subset B of X. In order to investigate what happens when we iterate the mapping W, it is necessary to restrict the sets B. Let

$$\mathcal{H}(X) = \{K \subset X : K \text{ is nonempty and compact}\}.$$

We endow $\mathcal{H}(X)$ with the *Hausdorff metric* h defined by

$$h(A, B) = \max \left\{ \sup_{a \in A} \inf_{b \in B} d(a, b), \sup_{b \in B} \inf_{a \in A} d(a, b) \right\}.$$

To get a feeling for the meaning of $h(A, B)$, notice that if $h(A, B) \leq \epsilon$, then every point of A is within ϵ of some point of B and conversely. It is possible to prove that $(\mathcal{H}(X), h)$ is a complete metric space (see [**1**]).

Let $\mathcal{W} = \{w_i\}$ be an IFS with contraction factor s. Then clearly $W(B)$ belongs to $\mathcal{H}(X)$ whenever B does, so $W : \mathcal{H}(X) \to \mathcal{H}(X)$. Furthermore, the map W is a contraction with contraction factor s (see [**1**]). Thus, by the Contraction Mapping Theorem, W has a unique fixed point A in $\mathcal{H}(X)$; i.e., A satisfies the condition that

$$A = W(A) = \bigcup_i w_i(A).$$

We call the fixed point the *attractor* of the IFS. For our motivating example, the Sierpiński gasket is the attractor of the three-map IFS given earlier.

How do we generate an image of A? Well, since W is contractive, we can start with *any* initial member A_0 of $\mathcal{H}(X)$ and form the sequence of sets given recursively by $A_{n+1} = W(A_n)$: this sequence converges to the attractor A. This is the so-called deterministic method of rendering an attractor and is illustrated in Figure 2, where we start with an initial image that is the attractor of another IFS.

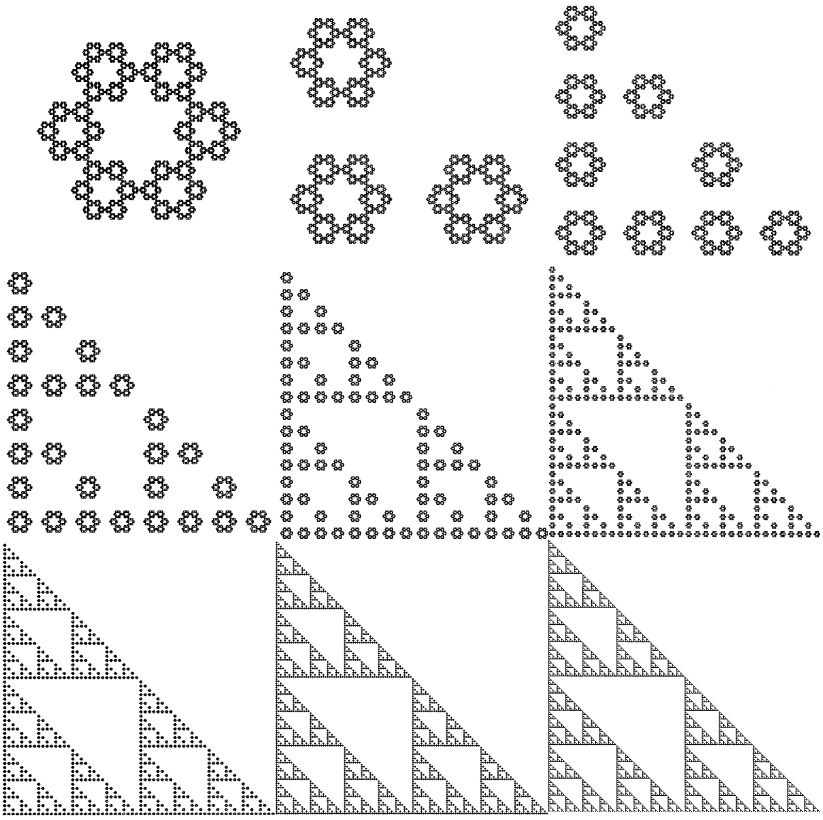

Figure 2. Deterministic iteration for the Sierpiński gasket.

Another method of rendering an image of the attractor of the N-map IFS $\mathcal{W} = \{w_i\}$ is the *chaos game algorithm*, which has the following steps:

1. Choose some point x_0 of X.
2. Choose a map w_i uniformly at random from the maps in the IFS $\mathcal{W}$ (that is, choose randomly from the set $\{w_1, w_2, \ldots, w_N\}$, with each choice equally likely).
3. Let $x_{n+1} = w_i(x_n)$ and plot the point x_{n+1}.
4. Return to step 2 and repeat the process, with x_{n+1} replacing x_n; iterate this procedure until "sufficiently" many points are generated.

Usually we let the algorithm run for some number of iterations before we start plotting points. It is a remarkable fact that an image of the set A "coalesces" from the set of plotted points as the iteration progresses. To see why this might be so, we reflect on what happens as we iterate this process.

Going back to our example IFS for the Sierpiński gasket, suppose that we start with the two points $x_0 = (0, 0)$ and $y_0 = (1, 1)$, and that we choose map w_0 in the first iteration of the chaos game. Then we have $x_1 = w_0(x_0) = (0, 0)$ and $y_1 = w_0(y_0) = (1/2, 1/2)$, so the distance between x_0 and y_0 has decreased by a factor of two. We see that, with each additional map that we apply to *both* x_n and y_n, distance is decreased by a factor of two. After a certain number of iterations, the two points x_n and y_n will be so close that we won't be able to distinguish them, at least not at the finite resolution of a computer screen. For example, if the screen has resolution 1024×1024, then after at

most eleven iterations, the two points that started out at opposite corners of the screen will have been contracted down to within one pixel, with the consequence that both will be plotted as a single pixel.

What does this have to do with the chaos game? Well, suppose that in the deterministic iteration of the set function W we start with the set $A_0 = \{x\}$, which represents a single pixel on the computer screen. Then

$$\begin{aligned} A_1 &= \{w_0(x), w_1(x), w_2(x)\} \\ A_2 &= \{w_0(w_0(x)), w_0(w_1(x)), w_0(w_2(x)), w_1(w_0(x)), \\ &\quad w_1(w_1(x)), w_1(w_2(x)), w_2(w_0(x)), w_2(w_1(x)), w_2(w_2(x))\}, \end{aligned}$$

and so forth. We find that

$$A_n = \{w_{i_1} \circ w_{i_2} \circ \cdots \circ w_{i_n}(x) : i_j \in \{0, 1, 2\}\}.$$

Thus, the chaos game is finding *one* such sequence of compositions, whereas the deterministic iteration generates all possible sequences. Now, by what we mentioned earlier in regard to the Sierpiński gasket on our 1024×1024 resolution screen, we cannot distinguish between the two points

$$w_{i_1} \circ w_{i_2} \circ \cdots \circ w_{i_n}(x), \quad w_{i_1} \circ w_{i_2} \circ \cdots \circ w_{i_n}(y)$$

once n reaches eleven, no matter what the points x and y are. (Notice that the two sequences of maps that we applied to x and y are the same; this is important.) As a result, in our exhaustive generation of such possible sequences, all we need to do is generate those of length at most eleven, since parts obtained from longer words in the alphabet $\mathcal{W}$ are visually indistinguishable.

Upon further reflection, we see that this gives a convenient way of labeling the points of the set A (at least on our limited resolution computer monitor) by using finite sequences of maps chosen from the IFS $\mathcal{W}$. To each such sequence of compositions of maps, there corresponds one point of the attractor A. However, it is possible that two such sequences of compositions of maps lead to the same point.

So where does this leave us? We see that in order to render an image of the Sierpiński gasket we need generate only the set of all possible sequences of length eleven from the set $\{0, 1, 2\}$, apply the appropriate compositions to ANY starting point, and plot the result. There are exactly $3^{11} = 177{,}147$ such sequences, meaning that there should be this many points on the approximate Sierpiński gasket on our 1024×1024 resolution computer screen (the screen has 1,048,576 pixels, a number considerably larger than 177,147).

Now we have the facts necessary to give us an inkling of why the chaos game works. When we generate the random sequence of maps to compose, with probability one we will *eventually* generate every substring of length eleven as a part of the sequence. Suppose, for instance, that part of a randomly generated string is

$$\ldots 0012122110021 \mid 01101020102212101 00 \ldots$$

and we have gotten as far as the | mark. Then the fact that the first three (displayed) maps are w_0, w_0, and w_1 makes no difference, since regardless of what point we start with, after eleven iterations we have lost any means of discovering where we started. Thus, only the most recent eleven iterations can have any influence on which pixel is

plotted. Since we generate all possible substrings of length eleven from {0, 1, 2}, we will eventually plot every point on the attractor.

This leads us to wonder if it is possible to do this more efficiently. Surely the chaos game algorithm will plot the same point multiple times. If there were some way of ensuring that we generate each point (or each sequence) only once, then this could save some effort, not to mention computation time.

3. DE BRUIJN SEQUENCES. We finished the previous section expressing the desire to find a single sequence from {0, 1, 2} that contains each sequence from {0, 1, 2} of a given length l as a subsequence exactly once. Such sequences do indeed exist; for example, for length $l = 2$ we have the sequence

$$0011221020$$

from which we get all the subsequences

$$00, 01, 11, 12, 22, 21, 10, 02, 20$$

by taking a "window" of length two and sliding it along the original sequence. If we wrap around on our sequence (say, by labeling evenly spaced points on a circle), we can shorten the "universal" sequence for $l = 2$ to

$$001122102.$$

Such wrap-around sequences are called de Bruijn sequences (see [**3**], [**4**], and [**6**]). Formally, a *de Bruijn sequence* corresponding to positive integers l and N is a circular sequence of length N^l such that every string of length l on an alphabet $\{a_1, a_2, \ldots, a_N\}$ occurs exactly once as a contiguous substring. The questions are: *Do such sequences exist for an arbitrary "window" length and an arbitrary number of symbols? If such sequences exist, how do we construct them?*. The remarkable fact is that we can find a simple algorithm to construct such sequences, thereby answering both questions at the same time! In this section we construct de Bruijn sequences using a tiny bit of graph theory, and in the next section we show how we can also exploit some of the theory of finite fields to construct de Bruijn sequences.

We illustrate the general algorithm by seeing how it works for the particular example we have been discussing. Consider the directed graph with three vertices labeled 0, 1, and 2. Draw directed edges in both directions from each vertex to each other vertex, including a single directed edge from a vertex to itself (see Figure 3). Label each edge by the name of the vertex that it leads to. This graph is an example of a *de Bruijn graph* (the definition of the de Bruijn graph for the general situation will be given shortly).

We wish to find an *Eulerian cycle* in this graph, by which we mean a path that traces out each edge exactly once and finishes where it starts. It is easy to see that in a directed graph such a cycle exists if and only if the number of edges leading out of each vertex (its out-degree) is the same as the number of edges leading into that vertex (its in-degree). This is clearly the case with our graph.

The Eulerian cycle that generates the sequence in our illustrative example is obtained by starting at vertex 0, then making the following sequence of moves (where we list the name of the edge that we traverse):

$$011221020.$$

Notice that this is our de Bruijn sequence.

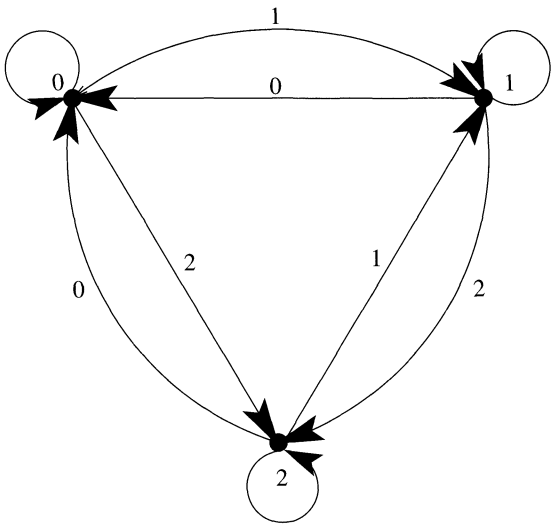

Figure 3. Directed graph for a de Bruijn sequence.

Suppose that we have the symbol set $\{0, 1, \ldots, N-1\}$ and we wish to find a single sequence with the feature that, as we slide a "window" of length l along this sequence we will obtain all possible sequences of length l, each exactly once, constructible from our symbol set. Clearly our long sequence must have length at least N^l elements (since we need at least that many different positions of the sliding window). It is an amazing fact that it is possible to find (many) sequences of exactly this length that will work.

The graph theoretic construction will work in this more general context as well. As our vertex set, we use the set of all words (sequences) of length $l-1$ from the alphabet $\{0, 1, \ldots, N-1\}$. We construct the edges of our directed graph by connecting vertex $a_1a_2 \ldots a_{l-2}a_{l-1}$ to vertex $a_2a_3 \ldots a_{l-1}b$ with a directed edge labeled b, for each choice of b. Again, the in-degree is equal to the out-degree for each vertex, so Eulerian cycles exist in this directed graph. Each such cycle corresponds to a de Bruijn sequence.

It is simple to implement a computer algorithm for finding an Eulerian cycle in a de Bruijn graph. First, define an integer array `vertex[]`. This array is indexed by all the possible vertices in the de Bruijn graph. Since there are N^{l-1} such vertices, the array `vertex[]` contains this many memory locations. In order to index the array, we interpret vertex $a_1a_2 \ldots a_{n-1}$ as a base N number (so `vertex[`$000\ldots0$`]` is the first memory location in the array). We initialize the array so that each entry has value $N-1$. Then we choose one vertex (say the vertex $000\ldots0$) as the starting point of the cycle, and whenever we are in a vertex j, we leave this vertex along the edge `vertex[j]` and decrease the value `vertex[j]` by one. That is, the array `vertex[j]` keeps track of which outgoing edges are still unused whenever we come to vertex j. We just use the outgoing edges in the order $N-1, N-2, N-3, \ldots, 1, 0$ to make the bookkeeping simple.

If the algorithm returns to a vertex with no remaining outgoing edges, check to see if it has generated the entire sequence (whose length is known). If not, find some vertex whose outgoing edges have not been exhausted and, by the same process, generate a new sequence to be "inserted" into the original sequence at the appropriate point. Continue this procedure until all edges have been accounted for. In the end, we will have generated the entire de Bruijn sequence and can simply print it out.

As an example, in the de Bruijn graph of Figure 3, if we start at vertex 0 we take the outgoing edge 2 to come to vertex 2. Then the procedure would take the sequence

of edges 21201100 to produce the de Bruijn sequence

$$221201100.$$

In this simple example, the algorithm never needed to "insert" a new sequence into the original sequence.

Suppose that we do this for the symbol set $\{0, 1, 2\}$ for length $l = 11$ and obtain our "mother" string of length $3^{11} = 177,147$. In order to render an image of the Sierpiński gasket we use this sequence instead of the randomly generated sequence of 0s,1s, and 2s. In this way, we are guaranteed to plot each point of the attractor exactly once, modulo the resolution of our 1024×1024 screen.

Notice that we can think of the chaos game as a random walk on the appropriate de Bruijn graph. In order for the chaos game to generate all points of the attractor (to some specified resolution), we need to wait until this random walk has traversed every edge in the graph.

A problem with this method is that we need to store the entire sequence in order to use it. Both the algorithm that generates the sequence and the modified "chaos game" require the entire sequence, so both algorithms require huge amounts of computer memory. It would be very nice if there were a way to generate the sequence as it is needed, rather than all at once.

4. CONSTRUCTION VIA FINITE FIELDS. We begin once more with an example to illustrate the main idea. Suppose again that we wish to find a de Bruijn sequence for the symbol set $\{0, 1, 2\}$ and a window of length $l = 2$. Consider the polynomial $q(x) = x^2 + x + 2$ as a polynomial over $\mathbb{Z}_3$ (the integers modulo 3). Starting with the polynomial $h(x) = x$ we recursively define a sequence of polynomials $h_0(x)$, $h_1(x)$, $h_2(x)$, ..., (over $\mathbb{Z}_3$) as follows: $h_0(x) = h(x)$ and $h_{n+1}(x)$ is obtained from $h_n(x)$ by multiplying the latter by x and reducing modulo $q(x)$ whenever necessary (that is, by exploiting the relation $q(x) \equiv 0$ or $x^2 = 2x + 1$ to eliminate powers of x higher than the first power). Doing this, we generate the sequence of polynomials (over $\mathbb{Z}_3$)

$$x, 2x + 1, 2x + 2, 2, 2x, x + 2, x + 1, 1,$$

at which point the iteration repeats itself (for example,

$$x(2x + 1) = 2x^2 + x = 2(2x + 1) + x = 5x + 2 = 2x + 2$$

in $\mathbb{Z}_3[x]/\langle q(x)\rangle$). Now choose a degree, take the coefficient of the term of this degree in each of these polynomials (say the coefficient of x in each), and write them down in sequence:

$$12202110.$$

Notice that we almost have a de Bruijn sequence corresponding to the given data. The only sequence of length two we are missing is 00. If we start out our sequence with a leading zero, we get the de Bruijn sequence

$$012202110.$$

If we had chosen to write down the constant coefficients (with the leading 0) we would obtain the de Bruijn sequence

How does this work? To explain it, we briefly review the basics of the theory of finite fields (the book [**2**] has a nice discussion of the necessary background). Recall that for any prime p the ring of integers modulo p, signified by $\mathbb{Z}_p$, is a field (since it has no zero divisors). Let

$$q(x) = x^l - a_{l-1}x^{l-1} - a_{l-2}x^{l-2} - \cdots - a_1x - a_0$$

be a polynomial of degree l in $\mathbb{Z}_p[x]$ that is irreducible over $\mathbb{Z}_p$, and consider the quotient ring

$$\mathbb{Z}_p[x]/\langle q(x)\rangle$$

(that is, the quotient ring of $\mathbb{Z}_p[x]$ modulo the ideal generated by the polynomial $q(x)$). We know, since $q(x)$ is irreducible, that $\mathbb{Z}_p[x]/\langle q(x)\rangle$ is a field with exactly p^l elements; it is called the *Galois field of order* p^l and often denoted by $GF(p^l)$. Addition in this field amounts to addition in $\mathbb{Z}_p[x]$. It is only in multiplying that we need reduce modulo $q(x)$. We do this by appealing to the relation

$$x^l = a_{l-1}x^{l-1} + a_{l-2}x^{l-2} + \cdots + a_1x + a_0.$$

Clearly we can view $GF(p^l)$ as a vector space of dimension l over $\mathbb{Z}_p$ (since $\mathbb{Z}_p[x]$ is clearly a vector space over $\mathbb{Z}_p$ and this field is a quotient vector space thereof). Furthermore, the field $GF(p^l)$ is the unique field with p^l elements and contains all roots of $q(x)$ as a polynomial in $Z_p[x]$.

Because $GF(p^l)$ is a field, its set of nonzero elements, the set $GF(p^l)^*$, forms an Abelian group under multiplication–in fact, a cyclic group. Suppose that we choose our polynomial $q(x)$ in such a way that one of its roots α in $GF(p^l)$ is a generator of this cyclic group: $GF(p^l)^* = \{\alpha, \alpha^2, \ldots, \alpha^{p^l-1}\}$ (such a root is called a *primitive root*).

This is what we did in our example, where $p = 3, l = 2$, and $q(x) = x^2 + x + 2$. Since all elements of $GF(3^2)$ can be viewed as polynomials of degree at most 1, we choose the polynomial so that $\alpha = x$ is a primitive root (notice that our notation does not distinguish between x in $\mathbb{Z}_p[x]$ and $x + \langle q(x)\rangle$ in $\mathbb{Z}_p[x]/\langle q(x)\rangle$). We see that the powers of x do indeed generate $GF(3^2)^*$, for there are eight elements in $GF(3^2)^*$ and we generated eight different polynomials by repeatedly multiplying by x.

What did we do when we chose one of the coefficients to write down? Well, $GF(3^2)$ is a vector space over $\mathbb{Z}_3$, so taking either coefficient can be thought of as computing the value of a linear functional $\phi : GF(3^2) \to \mathbb{Z}_3$. Thus we enumerated all the elements of $GF(3^2)^*$ and applied the linear functional ϕ to each to obtain our desired sequence. Notice that, if we had selected the linear functional that simply adds the two coefficients, we would have arrived at

$$010122021,$$

from which we obtain a de Bruijn sequence by inserting a 0 at the appropriate place.

It seems that all we are doing is generating all the elements of $GF(p^l)$ in a particular order. How do we know that we will get a de Bruijn sequence? To answer this question, suppose that

$$q(x) = x^l - a_{l-1}x^{l-1} - a_{l-2}x^{l-2} - \cdots - a_1x - a_0$$

is irreducible over $\mathbb{Z}_p$ and that $\alpha = x$ is a primitive root of $q(x)$ in $GF(p^l) = \mathbb{Z}_p[x]/\langle q(x)\rangle$. Let $\phi : GF(p^l) \to \mathbb{Z}_p$ be any nonzero linear functional (linear when we view $GF(p^l)$ as a vector space over $\mathbb{Z}_p$). Writing $GF(p^l)^* = \{\alpha, \alpha^2, \ldots, \alpha^{p^l-1}\}$ and applying ϕ to its elements in this prescribed order, we generate the sequence

$$\phi(\alpha), \phi(\alpha^2), \phi(\alpha^3), \ldots, \phi(\alpha^{p^l-1}).$$

Assume that there is a repeat in the window of length l at some point. Then there exist i and j satisfying $1 \le i < j \le p^l - 1$ such that

$$\begin{aligned} \phi(\alpha^i) &= \phi(\alpha^j), \\ \phi(\alpha^{i+1}) &= \phi(\alpha^{j+1}), \\ &\vdots \qquad\qquad \vdots \\ \phi(\alpha^{i+l-1}) &= \phi(\alpha^{j+l-1}). \end{aligned}$$

However, this is the same thing as saying that the l vectors $\alpha^{j+k} - \alpha^{i+k}$ for $k = 0, \ldots, l-1$ are in the kernel of the linear functional ϕ. Since $GF(p^l)$ is l-dimensional, these vectors must be linearly dependent; i.e., there are constants a_k in $\mathbb{Z}_p$, not all 0, for which

$$\sum_{k=0}^{l-1} a_k(\alpha^{j+k} - \alpha^{i+k}) = 0$$

or, equivalently,

$$\alpha^i(1 - \alpha^{j-i}) \sum_{k=0}^{l-1} a_k\alpha^k = 0.$$

Now $\alpha^i \neq 0$ and the only way to have $1 = \alpha^{j-i}$ is to take $i = j$, so it must be the case that $\sum a_k\alpha^k = 0$. However, this cannot happen, since then $g(\alpha) = 0$ for a polynomial $g(x)$ of degree $l-1$, contradicting the assumption that α is algebraic of degree l over $\mathbb{Z}_p$. Thus, no repeated windows are possible.

Notice that this argument also works if there is some "wrap around" in the subsequence. That is, it makes no difference to the argument if $j + l - 1 > p^l - 1$ (since the powers of α also "wrap around").

The reason we need to insert a zero into the sequence

$$\phi(\alpha), \phi(\alpha^2), \phi(\alpha^3), \ldots, \phi(\alpha^{p^l-1})$$

in order to produce a de Bruijn sequence is that we will not generate 0 in $GF(p^l)$ by this process. To remedy the situation, all we need to do is to find some place where there are $l-1$ zeros and insert an extra zero. By choosing ϕ to be the coefficient of x^{l-1}, we can simply prefix the zero, since in this instance the sequence starts with $l-1$ zeros.

What we are doing is using the multiplicative structure of $GF(p^l)$ to generate the $p^l - 1$ different elements of $GF(p^l)^*$ in a specific order and then using the linear functional ϕ to translate this information into a symbol from $\mathbb{Z}_p$.

Fast algorithm to compute de Bruijn sequences. It is easy to describe a fast algorithm to compute a de Bruijn sequence for a prime number of symbols (see chapter 2

of [**4**] for more details). For the irreducible polynomial $q(x)$ in the foregoing discussion, we use the recursion satisfying the initial conditions $y_{l-1} = 1$ and $y_i = 0$ for $i = 0, 1, \ldots, l-2$ with the recursion rule

$$y_{n+l} = a_{l-1}y_{n+l-1} + a_{l-2}y_{n+l-2} + \cdots + a_1 y_{n+1} + a_0 y_n$$

for $n = 0, 1, 2, \ldots$. We again need to insert a zero in the appropriate place. However, because of the initial conditions, we know that we can simply place the extra zero at the beginning.

Composite *n*. Now clearly this method of using finite fields is very nice, as it allows us to generate the de Bruijn sequence on the fly (if we know the polynomial $q(x)$). However, what if we wish to generate a de Bruijn sequence with a window length of l for the set $\{0, 1, \ldots, n-1\}$, where n is not a prime? What then? The problem is that in general $\mathbb{Z}_n$ is not a field and there is no associated field $GF(n^l)$.

Case 1. $n = p^k$ for a prime p and $k > 1$.

In this situation we exploit the fact that $GF(p^k)$ is a subfield of $GF(p^{kl})$, making $GF(p^{kl})$ a vector space over $GF(p^k)$. What we do is to use a linear functional $\phi : GF(p^{kl}) \to GF(p^k)$. Our set of "symbols" will be from $GF(p^k)$, which we can then interpret as the set $\{0, 1, 2, \ldots, p^k\}$.

An example will help to make this clear. Suppose that $n = 4$ and $l = 2$. In general, finding a primitive polynomial of degree l over $\mathbb{Z}_p$ is difficult. In our case, since both l and p are small, we simply used `MAPLE` to factor the polynomial $z^{2^4} - z$ over $\mathbb{Z}_2$ and picked $q(z) = z^4 + z + 1$ as one of the factors of degree 4. Since $q(z)$ does not divide $z^m - z$ for any m smaller than 16, we know that it must be primitive (for this and other results related to finding irreducible or primitive polynomials, see [**4**]). We find that the polynomial $q(x) = x^4 + x + 1$ is irreducible over $\mathbb{Z}_2$ with $\alpha = x$ being a primitive root in $\mathbb{Z}_2[x]/\langle q(x)\rangle$. We proceed as we did before, computing the powers of α in order to generate $GF(2^4)^*$.

Now we need to come up with a linear functional $\phi : GF(2^4) \to GF(2^2)$. To do this, notice that $GF(2^2)$ is the subset $\{0, \alpha^5, \alpha^{10}, \alpha^{15} = 1\}$ of $GF(2^4)$, where $\alpha^5 + 1 = \alpha^{10}$. We know that all these elements of $GF(2^4)$ are in $GF(2^2)$ because they are all roots of the polynomial $z^2 + z + 1$, which generates $GF(2^2)$ over $\mathbb{Z}_2$. If we can represent a given β in $GF(2^4)$ in some basis over $GF(2^2)$, then we can simply take a component of this representation as our linear functional ϕ (as we did earlier by taking the coefficient of x^{l-1}).

Given an element β of $GF(2^4)$, how do we find its representation? The obvious basis of $GF(2^4)$ over $GF(2^2)$ is the set $\{\alpha, 1\}$. Next we observe that

$$\begin{aligned} 1 &= 0 \cdot \alpha + 1 \cdot 1, \\ x &= 1 \cdot \alpha + 0 \cdot 1, \\ x^2 &= 1 \cdot \alpha + \alpha^5 \cdot 1, \\ x^3 &= \alpha^{10} \cdot \alpha + \alpha^5 \cdot 1, \end{aligned}$$

so an arbitrary element $\beta = a + bx + cx^2 + dx^3$ of $GF(2^4)$ (using our representation of $GF(2^4) = \mathbb{Z}_2[x]/\langle q(x)\rangle$) can be expressed as $\beta = A \cdot \alpha + B \cdot 1$, with

$$\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & \alpha^{10} \\ 1 & 0 & \alpha^5 & \alpha^5 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix},$$

all arithmetic being carried out in $GF(2^4)$. Thus we can take our linear functional ϕ to be represented by the row-vector

$$\begin{pmatrix} 0 & 1 & 1 & \alpha^{10} \end{pmatrix}.$$

Putting all this together, the sequence of powers of $\alpha = x$ is:

$$x, x^2, x^3, x+1, x^2+x, x^3+x^2, x^3+x+1, x^2+1,$$
$$x^3+x, x^2+x+1, x^3+x^2+x, x^3+x^2+x+1, x^3+x^2+1, x^3+1, 1.$$

Applying ϕ to this sequence we get the sequence (in $GF(2^2)$)

$$1, 1, \alpha^{10}, 1, 0, \alpha^5, \alpha^5, 1, \alpha^5, 0, \alpha^{10}, \alpha^{10}, \alpha^5, \alpha^{10}, 0.$$

Finally, we can use any mapping $GF(2^2) \to \{0, 1, 2, 3\}$ and judiciously insert a 0 to retrieve the de Bruijn sequence

$$1131022120332300$$

(here we attached the extra 0 at the end).

We mention in passing that in [**4**] there is a discussion of the representation of all linear functionals $\phi : GF(p^{kl}) \to GF(p^k)$ using a "trace"

$$Tr(\beta) = \beta + \beta^q + \beta^{q^2} + \cdots + \beta^{q^{l-1}},$$

where $q = p^k$.

Case 2. n is not a power of a prime.

What happens if n is not a prime power? Suppose, for instance, that $n = 6 = 2 \cdot 3$ and $l = 2$. We take a de Bruijn sequence for $n = 2$ and $l = 2$, say

$$0110,$$

and a de Bruijn sequence for $n = 3$ and $l = 2$, say

$$001220211,$$

then repeat the first one nine times and the second four times to get

$$011001100110011001100110011001100110$$
$$001220211001220211001220211001220211.$$

Taking in order one symbol from the top line and the one below it from the bottom, and using the identifications

$$00 \mapsto 0, 01 \mapsto 1, 02 \mapsto 2, 10 \mapsto 3, 11 \mapsto 4, 12 \mapsto 5,$$

we obtain

$$034223511331253214301550244004520541.$$

Thus by "mixing" two de Bruijn sequences for 2 and 3 we get one for 6.

In the general case suppose that the prime factorization of n is

$$n = p_1^{m_1} \cdot p_2^{m_2} \cdots p_k^{m_k}$$

with $p_1 < p_2 < p_3 < \cdots < p_k$ and that we wish to obtain a sequence with a window of length l. By the Chinese Remainder Theorem, $\mathbb{Z}_n \approx \mathbb{Z}_{p_1^{m_1}} \times \mathbb{Z}_{p_2^{m_2}} \times \cdots \times \mathbb{Z}_{p_k^{m_k}}$. Let Θ be an isomorphism from the indicated product group to $\mathbb{Z}_n$. Our construction using $GF(p_k^{(m_k)l})$ as a vector space over $GF(p_k^{m_k})$ yields k de Bruijn sequences $\{v_j^i\}$ for $i = 1, \ldots, k$ and $j = 1, \ldots, p_i^{m_i l}$, with v_j^i in $\mathbb{Z}_{p_i{}^{m_i}}$. We obtain the desired de Bruijn sequence (in $\mathbb{Z}_n$) by applying Θ to the combination of these sequences. Specifically, for $r = 1, \ldots, n^l$ the rth term of the sequence is

$$w_r = \Theta\left(v^1_{r \bmod p_1^{m_1}}, \ldots, v^k_{r \bmod p_k^{m_k}}\right).$$

We claim that this sequence has the desired property. Clearly each term of the sequence belongs to $\mathbb{Z}_n$. Suppose that there were a repeated block of length l. Then for some a and b with $a < b$ we would have

$$\begin{aligned} w_a &= w_b, \\ w_{a+1} &= w_{b+1}, \\ &\vdots \\ w_{a+l-1} &= w_{b+l-1}. \end{aligned}$$

Since Θ is an isomorphism, this would imply that

$$\begin{aligned} v^i_{a \bmod p_i^{m_i}} &= v^i_{b \bmod p_i^{m_i}}, \\ v^i_{a+1 \bmod p_i^{m_i}} &= v^i_{b+1 \bmod p_i^{m_i}}, \\ &\vdots \\ v^i_{a+l-1 \bmod p_i^{m_i}} &= v^i_{b+l-1 \bmod p_i^{m_i}}. \end{aligned}$$

for $i = 1, 2, \ldots, k$. This cannot happen for all i simultaneously by the de Bruijn property of the individual sequences $\{v_j^i\}$ and the fact that the primes p_i are distinct. Again this argument also works if there is some "wrap around" in the subsequences.

Basically we "mix" together de Bruijn sequences for relatively prime periods and relatively prime symbol spaces by using the Chinese Remainder Theorem essentially twice–once to get all the symbols in the symbol space and again to prove that the sequence we obtain has no repeats.

5. BACK TO FRACTALS. How efficient is it to use the finite field method for constructing de Bruijn sequences in rendering a fractal? The main difficulty is in finding the primitive polynomials (those irreducible polynomials whose roots generate $GF(p^l)^*$) and, in the case $n = p^k$, a representation for the linear functional ϕ. If we are willing to hard-code this information into our program, then the algorithm is very efficient.

Oftentimes it is merely necessary to get a reasonable image of the attractor of an IFS. In these cases, running the chaos game with a very low iteration count (or even a short de Bruijn sequence) is sufficient. However, there are cases where it is essential that we guarantee that every point of the attractor (at the specified resolution) is plotted (for example, in situations where we wish to analyze the geometrical structure of the attractor in some automatic way). Under such circumstances the de Bruijn sequence approach is much more efficient than the chaos game since we know that the former will generate all the relevant points. With the chaos game we could run a very high iteration count, but we would still have only probabilistic bounds. For example, we expect that it will take roughly 2.24 million iterations of the chaos game in order to get all possible length eleven sequences from the set $\{0, 1, 2\}$.

Finally, we wish to mention that there is a wealth of literature on using de Bruijn sequences in many application areas. If the reader is interested in exploring this topic, a good way to start would be to go to google (or one's favorite search engine) and do a keyword search. The book [**4**] has an extensive bibliography and contains a chapter on linear recurrence sequences in $GF(p^l)$. The book [**5**] by the same authors is more of a textbook and provides a very nice introduction to the theory of finite fields.

ACKNOWLEDGMENTS. The author appreciates discussions with Juaquin Anderson, who suggested the idea of using de Bruijn sequences to drive a modified chaos game. The author would also like to thank Tom Archibald for his helpful comments and criticisms and Neil Calkin for pointing out the references [**4**] and [**5**]. This work was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

1. M. G. Barnsley, *Fractals Everywhere*, Academic Press, New York, 1988.
2. R. A. Dean, *Classical Abstract Algebra*, Harper and Row, New York, 1990.
3. N. G. de Bruijn, A combinatorial problem, *Indag. Math.* **8** (1946) 461–467.
4. R. Lidl and H. Niederreiter, *Finite Fields*, Encyclopedia of Mathematics and Its Applications, vol. 20, Addison-Wesley, Reading, MA, 1983.
5. ———, *Introduction to Finite Fields and Their Applications*, Cambridge University Press, Cambridge, 1994.
6. S. W. Golomb *Shift Register Sequences*, Holden-Day, San Francisco, 1967.

FRANKLIN MENDIVIL is an assistant professor of mathematics at Acadia University in Nova Scotia. His research is a mixture of fractal geometry and analysis, topology, image processing, and optimization. He considers himself extremely lucky to be in a profession that allows him to explore many different topics.
Acadia University, Wolfville, Nova Scotia, B4P 2R6, CANADA
franklin.mendivil@acadiau.ca