A Fractal Representation for Real Optimization

Daniel Ashlock
Department of Mathematics and Statistics
University of Guelph
Guelph, Ontario, Canada, N1G 2W1
dashlock@uoguelph.ca

Justin Schonfeld
Department of Computer Science and Engineering
University of Nevada at Reno
Reno, NV, 89557, USA
shonfju@cse.unr.edu

Abstract—The chaos game, in which a moving point is repeatedly averaged toward randomly selected vertices of a triangle, is one method of generating the fractal called the Sierpinski triangle. The sequence of vertices, called generators, used to reach a given point of the Sierpinski triangle yields a map from strings over a three-character alphabet to points in the plane. This study generalizes that representation to give a character-string representation for points in \( \mathbb{R}^n \). This is a novel representation for evolutionary optimization. With the correct generating points the method is proven to search its entire target domain at an easily controlled resolution. The representation can be used to achieve the same goals as niche specialization at a far lower computational cost because the optima located are specified by strings which can be stored and searched in standard string dictionaries. An implementation of the algorithm called the multiple optima Sierpinski searcher (MOSS) is found to be substantially faster at locating diverse collections of optima than a standard optimizer. The Sierpinski representation has a number of natural mathematical properties that are described in the paper. These include the ability to adapt both it search domain and its resolution on the fly during optimization.

I. INTRODUCTION

When designing an evolutionary computation system the issue of representation is key. The encoding of the solution can have a huge impact on time to solution, solution quality, and algorithm performance. This study introduces the Sierpinski representation for real-parameter optimization. This representation will seem complex and unnatural initially, but places a string structure on Euclidean space. This means that the encoding of the solution comes from the chaos game algorithm for generating the Sierpinski triangle shown in Figure 1. The chaos game is given in Algorithm 1.

Algorithm 1: The Chaos Game

**Input:** The vertices of a triangle in the real plane

**Output:** A set of points in the real plane

**Details:**
- Select one of the vertices \( V \) uniformly at random.
- Initialize a point \( P \) to the position of \( V \).
- Repeat
  - Select a vertex \( V \) uniformly at random;
  - Move \( P \) half-way to \( V \);
  - Plot \( P \).
- Until(Finished)

The chaos game uses a moving point initialized at one of the vertices of the triangle. It then iteratively chooses a vertex at random and moves half-way to the chosen vertex. Each of the points in the Sierpinski triangle is thus the result of a sequence of choices: the initial vertex and the sequence of vertices averaged toward while generating the point. This means that the points of the Sierpinski triangle are matched with the space of strings \( \{v_1, v_2, v_3\}^* \), where the \( v_i \) are the vertices of the triangle and the star superscript denotes “all finite strings over” in the usual manner. Henceforth we will call vertices generators and associate them with an alphabet over which a population strings is built. The simplest representation used in evolutionary computation is that of finite strings. Notice that the character strings of length \( n \) correspond to some subset of the Sierpinski triangle. This representation of some points in \( \mathbb{R}^2 \) by finite strings over a three letter alphabet can be generalized to a representation that is useful for optimization in \( \mathbb{R}^n \). The representation can be scaled for fine or coarse search in an almost trivial manner, presents optima as strings of characters that can be sorted and manipulated more easily than collections of points in \( \mathbb{R}^n \).

The remainder of this study is structured as follows. Section II gives the mathematical properties of the Sierpinski representation. This section is highly mathematical and proves many of the properties. Section III gives the design of the experiments used to test the representation and summarizes the results. Section IV discusses the results and draws conclusions. Section V outlines possible next steps for the research.

II. PROPERTIES OF THE SIERPINSKI REPRESENTATION

**Definition 1:** The Sierpinski representation. Let \( G = \{g_0, g_1, \ldots, g_{k-1}\} \) be a set of points in \( \mathbb{R}^n \) called the generator...
points for the Sierpinski representation. Associate each of the points, in order, with the alphabet \( A = \{0, 1, \ldots, k-1\} \). Let the positive integer \( r \) be the depth of representation. Then for \( s \in A^r \) let the point represented by \( s \), \( p_s \), be given by Algorithm 2.

**Algorithm 2: Sierpinski Unpacking Algorithm**  
**Input:** A string \( s \in A^r \)  
The set \( G \) of \( k \) generator points  
An averaging weight \( \alpha \)  
**Output:** A point in \( \mathbb{R}^n \)  
**Details:**  
Set \( x \leftarrow g_{s[r-1]} \)  
for \( i \leftarrow r - 2; i \geq 0; i \leftarrow i - 1 \)  
\( x \leftarrow \alpha \cdot g_{s[i]} + (1 - \alpha) \cdot x \)  
end for  
return(\( x \))

The Sierpinski representation for points in \( \mathbb{R}^n \) relative to the generator set \( G \) is their encoding by strings, given above. (Note that the string is unpacked from the end.)

For the Sierpinski triangle, \( k = 3 \) and \( \alpha = 0.5 \). In this case, the set of points that can be represented by any length of string have the unsatisfying property that, even within the interior of the triangle specified by the three vertices, there are large holes. If we were to evaluate only points in the Sierpinski triangle, then it would take a good deal of luck for our optima to be on (or acceptably near) representable points. Fortunately, additional generators can be proven to improve the situation to where the entire convex hull of the generators is being searched to an easily specified resolution. Recall that the convex hull of a set of points is a closure of the set under convex combinations. A convex combination of two points \( p \) and \( q \) is a linear combination of the form: \( \alpha \cdot p + (1 - \alpha) \cdot q \), \( 0 \leq \alpha \leq 1 \).

Define the set of \( r \)-representable points to be those that can be represented with strings of length at most \( r \) for a given set of generator points with the Sierpinski representation. Let the union, for all finite \( r \), of the \( r \)-representable points be the representable points. If \( S \) is a set of points in \( \mathbb{R}^n \) and \( T \subset S \) then we say that the covering radius \( R_T(S) \) of \( S \) by \( T \) is the largest distance that any point in \( S \) is from the point it is closest to in \( T \). In other words, \( S \) is a subset of the union of balls of radius \( R_T(S) \) about every point in \( T \). We say a subset \( T \subset S \) is dense if within every distance \( \epsilon > 0 \) of every point \( s \in S \) there is a point of \( T \). A dense subset may be thought of as having covering radius zero. A good explanation of dense subsets appears in [4].

**Theorem 1: Interval Density** Suppose we set the averaging parameter \( \alpha = 0.5 \). Suppose that we have two generators. Let \( S \) be the line segment whose ends are the generators. Then:

- the representable points are dense in \( S \), and
- the \( r \)-representable points have covering radius \( \frac{C}{2^r} \) where \( C \) is the length of the line segment \( S \).

**Proof:**

Assume for now that the line segment is the unit interval \([0,1]\) and that we start at 0. Let \( s \in \{0, 1\}^r \) be a Sierpinski representation on two generators. Then the point \( p_s \) is

\[
\sum_{i=1}^{r} \frac{s[i]}{2^i}.
\]

These numbers are the dyadic rationals, the rational numbers with denominator equal to a power of 2. At a given depth of representation \( r \) these points form a regular grid in the unit interval with spacing \( \frac{1}{2^r} \). The distributive law scales this spacing to \( \frac{1}{2^r} \), and the translation invariance of Euclidean distance on a set of points permits the result to hold for any line segment in \( \mathbb{R}^n \). This proves the claim about covering radius. The set of representable points contain subsets with covering radius \( \frac{C}{2} \) for any \( r \). Since there is some \( r \) so that \( \epsilon > \frac{C}{2} \) given fixed \( C \) it follows that the representable points are dense in the line segment \( S \).

Recall that an \( n \)-parallelepiped is the \( n \)-dimensional generalization of a parallelogram. The following corollary extends Theorem 1 to \( n \)-space. It may be helpful to think of the example of a square. If we take the corners of a square as our generators, then the \( x \) and \( y \) coordinates independently and simultaneously have the same representational dynamics as the line segment in Theorem 1. This is also true for \( n \)-cubes and even \( n \)-parallelepipeds, though the coverage is better in the shorter dimensions at a fixed depth of representation.

**Corollary 1: General Density** Suppose that \( \alpha = 0.5 \) and that the generators of a Sierpinski representation are the vertices of an \( n \)-parallelepiped \( R \). Then:

- the representable points are dense in \( R \), and
- the \( r \)-representable points have covering radius \( \frac{C}{2^r} \) in \( R \) where \( C \) is a constant.

**Proof:**

Any two opposite faces of \( R \) have \( n-1 \) coordinates which vary in the same way for both and one coordinate whose behavior separates them. In the dimension that separates these opposite faces, the proof of covering radius and density from Theorem 1 holds for the corresponding coordinate of \( n \) space. We thus achieve the desired result coordinate-wise for each coordinate. The value of \( C \) depends on the interplay of the
distances between various opposite faces of $R$. The fact that we obtain a two-fold improvement in the spacing of individual coordinates yields the two-fold improvement in the covering radius when the depth of representation is increased by one.

The density and covering radius results show that, so long as $\alpha = 0.5$ and the generator points form an $n$-parallelepiped, the Sierpinski representation searches the entire convex hull of the generator points at a resolution specified by the depth of representation. It is worth noting that in order to exploit the covering radius result $2^n$ generators are required for $n$-dimensions.

A. Unique Naming

Lemma 1: Let a string $s$ of length $r$ be a name for a point $x = p_s$. Suppose that $\alpha = 0.5$ and that the last character (initial generator) of $s$ is 0. Then $s$ is the sole name of $x$ of length $r$.

Proof:

Suppose that $s$ is a name for $x$ of length $r$. Then each character in $s$ specifies a digit of the base 2 expansion of the real number in each coordinate of $x$. Points that have coordinates with different digits are not equal, and the lemma follows. □

Definition 2: The normalized Sierpinski representation (NSR) is achieved by insisting that the last character of the string always be the first generator.

Notice that in an interval if we do not restrict the generator at which a point starts, then a representable point will have two names of each length as each representable point has a name starting with each generator. Notice also that every representable point in the normalized representation has an infinite number of names, each a different length. These infinitely many names are the result of “trailing zeros,” i.e. averaging toward the generator that the string ends with some number of times. Figure 2 shows the Sierpinski representations of length 0, 1, 2, and 3 for generators at the ends of the interval [0,1].

Lemma 1 says that the NSR with $\alpha = 0.5$ yields a 1:1 correspondence between strings of length $r$ and the points they represent. This unique naming result, in addition to being a nice property from the perspective of search, is convenient for another reason.

B. Dictionary Exclusion

Niche specialization [1] is a technique for encouraging exploration by penalizing a population that piles up near a given optimum. The goal is both to force the algorithm out of local optima and to encourage a broad survey of the search space. The Sierpinski representation is well-suited to broad search. Assume that $\alpha = 0.5$. Then the first character of a string contains 50% of the information about the location of the point it represents; the second character contains 25% of the representation; the third contains 12.5% and so on. This means that exclusion of areas already searched can be performed by looking at the initial segment the representation.

There are a number of ways that the details could be arranged; the following algorithm is tested in Section III.

Algorithm 3: Multiple Optima Sierpinski Searcher

Input: A set of generator points $G$, an averaging parameter $\alpha$, a depth of representation $r$, a depth of exclusion $d$, a multi-modal function $f$ to optimize

Output: A collection of optima

Details:

1. Initialize a population of Sierpinski representation strings.
2. Run a string-EA until an optimum $x$ is found.
3. Initialize a dictionary $D$ with the string specifying $x$.
4. Repeat
   - Re-run the EA, awarding minimal fitness to any string with the same $d$-prefix as any string in the dictionary.
   - Record the new optimum’s string in $D$.
5. Until(Enough optima are found).

The cost of niche specialization in an EA can be quite high because of the need to compute when a set of points are in the same optima and so require fitness penalization. Excluding optima from search that have already been located requires that optima be placed in some form of data base and compared, often in their entirety. The Multiple Optima Sierpinski Searcher (MOSS) algorithm permits exclusion of known optima, in a given run of an EA, at fixed algorithmic cost no matter how many optima have been located thus far. To perform niche specialization the population can be sorted into neighborhoods by a quick-sort on the Sierpinski strings; strings with identical prefixes are neighborhoods whose size is determined by the prefix size (henceforth called exclusion depth).

C. Search Refinement and Fixation

A smaller search space can be searched more quickly while a large search space contains a larger collection of candidate points. Suppose that the normalized Sierpinski representation with $\alpha = 0.5$ is being used for real parameter optimization. Suppose additionally that a 0 is added to the end of each string increasing the depth of representation $r$ by one. Then the fitness of each string remains unchanged while the search resolution has been increased two-fold. Recall that the strings are unpacked from the end and must terminate with a zero when normalized. Adding a zero to the end will make the first unpacking step a move from $g_0$ to itself: the final value of the unpacked point will be the same.

The process of adding a zero to the end of each member of a population is called neutral refinement and such adding of a zero is a form of neutral mutation. Once the depth of representation has been increased, the EA can refine its approximation of the true position of the optimum. This means that a Sierpinski representation EA can start with a small search space, rapidly locate a tentative optimum,
and then refine the search by simple increasing its depth of representation.

The string-based Sierpinski representation has another potentiality for refining search. Suppose that the prefixes of a population have, for the most part, converged. This indicates the localization of the search to a portion of the space specified by the common prefix. After such a common prefix has been located by evolution, it can be mandated by the EA. Fixation is the practice of recognizing a common prefix and forcing all population members to have that prefix. This has the effect of shrinking the search space to the area specified by the common prefix and, again, smaller spaces can be searched more rapidly.

A Sierpinski representation EA that used both refinement and fixation could rapidly find optima with very high numerical resolution. If the initial search was too coarse, this process could completely exclude some optima, and so the techniques should be used with care.

D. Refinement via Generators

While the string character of the representation yields the elegant properties that permit the control of search through rapid data-basing of known optima and easily implemented search refinement, the generators are also a point where user or programmer intervention can be used to control the search of an EA using the Sierpinski representation.

First note that the “coordinate axes” of search for the Sierpinski representation are defined by the line segments joining the generators and the distance between those generators. If the shape of a function being optimized does not agree well with the coordinate axes in which it is currently parameterized, then the search can be reparameterized by moving the generator points. If the good coordinatization for a given function is unknown, then the generator points could be subjected to random (or optimized) rotations before each run of the EA. This permits exploration of the coordinatization of the problem in a manner that yields benefits similar to those obtained with self-adaptation; moving the generator points is, however, a global adjustment while self-adaptation is local.

Second, note that by translating or carefully placing generator points we can search different portions of the space. If an initial search turns up a set of good points, then an EA with generators flanking that region can search in the immediate neighborhood of those points. Search can thus be refined by placement of generators. This sort of directed search may be especially valuable when attempting to locate collections of points such as the pareto-frontier for a multi-criterion optimization.

Finally we note that it is possible to restart a Sierpinski representation EA with added generators as a method of biasing the search. Suppose that an initial set of generators has located good points, but it is desirable to bias the search toward those parts of the search space without restricting it to their close neighborhood. Then the good points can be added as additional generators. This will increase the search density near those points without changing the actual boundaries of the search space. This follows from the fact that use of the additional generator moves a point toward that generator more often. Such addition of generators substantially changes the sampling density of the space and so should be used cautiously.

III. Experiments with the Representation

In order to understand the behavior of the Sierpinski representation for optimization, a series of experiments are performed on simple landscapes with various numbers of optima. The first is to find the optimum of

\[
f(x_0, x_1, \ldots, x_{n-1}) = \frac{0.8}{\sum_{i=0}^{n-1} (x_i - 13)^4 + 1} + \frac{1.0}{\sum_{i=0}^{n-1} (x_i - 5)^2 + 1}
\]

This equation has two optima, the lower of which has a much large basin of attraction in the search domain of \(0 \leq x_i \leq 21.6\). It is thus mildly deceptive, attracting an evolving population to the inferior optima.

The second experiment optimizes a function with eight distinct optima. Start with:

\[
g(x, y, z) = \frac{1}{1 + x^2 + y^2 + z^2}
\]

For each of the eight points:

| \(P_i\) | \{(16.0, 15.8, 11.8)\} |
| \(P_1\) | \{(12.4, 14.2, 9.0)\} |
| \(P_2\) | \{(15.5, 11.7, 11.4)\} |
| \(P_3\) | \{(18.6, 15.5, 16.0)\} |
| \(P_4\) | \{(3.9, 15.8, 10.1)\} |
| \(P_5\) | \{(12.6, 26.0, 1.0)\} |
| \(P_6\) | \{(17.8, 1.0, 10.7)\} |
| \(P_7\) | \{(1.8, 14.3, 18.1)\} |

set \(g_i(x, y, z) = g(x - P_{i_x}, y - P_{i_y}, z - P_{i_z})\) where \(P_{i_x}^x\), \(P_{i_y}^y\), and \(P_{i_z}^z\), refer to the \(x, y, \) and \(z\) coordinates of point \(i\) respectively. The second function to be optimized is

\[
M(x, y, z) = \sum_{i=0}^{7} g_i(x, y, z).
\]

The third experiment will be to find optima of the function:

\[
h(x_0, x_1, \ldots, x_{n-1}) = \sin(\sqrt{x_0^2 + x_1^2 + \cdots + x_{n-1}^2}) \cdot \prod_{i=0} \sin(x_i)
\]

This function possesses an infinite number of optima of varying heights and is thus good for testing the ability of an algorithm to locate a diversity of optima. Each of these functions is optimized in different ways with the details given in the section for each particular function.
A. Comparison and Parameter Tuning

This section deals with the optimization of the two-optima function used in the first experiment. It tests the Sierpinski representation in a simple context, compares it with a standard evolutionary algorithm, and is used for parameter setting. It is important to place the optimum somewhere other than in the sum of a small number of generators. In an initial set of runs the optimum was so placed and solution occurred in the initial population with high probability. In order to place the optimum away from small sums of generators, the generators are chosen to be the vertices of a hypercube with one corner at the origin and the opposite corner at \((a, a, \ldots, a)\) for \(a = 21.8\).

The problem is run in \(d = 3\) dimensions with a population size of \(n = 540\). The population size was chosen in a preliminary study to be in a region where both the Sierpinski and standard representations were functioning well so as to give an advantage to neither.

The evolutionary algorithm used to test the Sierpinski representation has the following parameters: population size, number of point mutations used, and depth of representation. The algorithm uses single tournament selection with tournament size seven. This is a steady-state model of evolution [5] where time is measured in mating events. In a mating event seven strings are chosen without replacement. The two most fit are copied over the two least fit. The copies undergo one-point crossover and mutation. The mutation replaces \(n\) characters, chosen with replacement, in the string with characters selected uniformly at random from the alphabet being used. The algorithm continues until 100,000 mating events have been made or until a fitness value of 0.99 is found. If such an acceptable fitness value is not located the algorithm is said to have timed out. Note that the true optimum of the function being optimized has a value of roughly 1.01. One-point crossover is used because almost all of the fitness of a string is determined by a prefix. This means that the edge effects avoided by using two-point crossover are swamped by the intrinsic bias in the representation. For each set of parameters the algorithm is run 1000 times, a 95% confidence interval on the time to solution is computed, and the number of timeouts is recorded. For comparison, a standard evolutionary algorithm using a real-array representation, size seven tournament selection, one-point crossover, and uniform real mutation of 1, 2, or 3 coordinates (adding a uniformly distributed random number with support \([\delta, \delta]\) to one coordinate) is used. The value of \(\delta\) was varied using \(\delta \in \{0.2, 0.5, 1.0\}\). The results are given in Table I. None of the runs timed out for either the fractal or standard representation.

B. Climbing Several Hills

When searching \(S\) to find its eight optima we use generators similar to those used with the single hill experiment but with \(a = 20\). The optima were selected at random within a \(20 \times 20 \times 20\) cube, and so there is no issue of the optima being lined up in a fortuitous manner with the generators. The experiment in this section will check to see if and how often the optima in question were located with the standard Sierpinski representation algorithm and the MOSS algorithm. The parameters for this algorithm are derived by interpolating the results of the single hill experiments. They are: population size 540, depth of representation \(r = 8\), and mutation that replaces 4 characters. The other parameters are left at the values used in the single hill experiments. The algorithm is run for 100,000 mating events, and the most fit population member is examined to see if it is one of the eight optima of Equation 2. Table II gives the results of running the Sierpinski representation evolutionary algorithm 1000 times. It locates the global optimum just over 85% of the time and locates only the three best optima of the eight available.

With multiple optima available it is now potentially profitable to test dictionary exclusion, and so the MOSS algorithm was implemented and applied to Equation 2. The Sierpinski representation EA used previously was modified as follows. Any Sierpinski string that agreed in its initial four characters with a known optimum was given a fitness of 0. The exclusion depth of 4 was chosen by looking at the number of fixed positions in the optima located in the 1000 initial runs. Table III shows a run trace for the first 12 populations run.

Only twelve runs of the MOSS algorithm were required to locate all eight optima. When an optimum was relocated the result was an inferior point within the optimum: such additional hits within an optimum were also recorded in the dictionary, increasing the zero-fitness region within the basin.
TABLE II
FREQUENCY OF LOCATION OF THE EIGHT OPTIMA FOR EQUATION 2 BY A SIERPINSKI REPRESENTATION OPTIMIZER WITH $r = 8$, TOGETHER WITH THE SIERPINSKI CODING AND THE FITNESS OF THE OPTIMA ASSOCIATED WITH THE EIGHT HILL CENTERS FOR EQUATION 2. STARS IN THE CODING INDICATE POSITIONS THAT VARIED BETWEEN DIFFERENT LOCATIONS OF THE OPTIMA.

<table>
<thead>
<tr>
<th>Point</th>
<th>Frequency</th>
<th>Encoding</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>851</td>
<td>6204****</td>
<td>1.153</td>
</tr>
<tr>
<td>$P_1$</td>
<td>2</td>
<td>2476125*</td>
<td>1.125</td>
</tr>
<tr>
<td>$P_2$</td>
<td>147</td>
<td>6107020*</td>
<td>1.146</td>
</tr>
<tr>
<td>$P_3$</td>
<td>0</td>
<td>n/a</td>
<td>1.086</td>
</tr>
<tr>
<td>$P_4$</td>
<td>0</td>
<td>n/a</td>
<td>1.050</td>
</tr>
<tr>
<td>$P_5$</td>
<td>0</td>
<td>n/a</td>
<td>1.048</td>
</tr>
<tr>
<td>$P_6$</td>
<td>0</td>
<td>n/a</td>
<td>1.045</td>
</tr>
<tr>
<td>$P_7$</td>
<td>0</td>
<td>n/a</td>
<td>1.032</td>
</tr>
</tbody>
</table>

TABLE III
BELOW ARE THE POPULATION-BEST RESULTS FROM THE FIRST 12 OF 100 RUNS OF THE MOSS ALGORITHM. STARS DENOTE THE FIRST TIME AN OPTIMA WAS LOCATED. THE LAST OF THE EIGHT AVAILABLE OPTIMA WAS LOCATED IN THE TWELFTH RUN.

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Functional Representation</th>
<th>Optima Located</th>
</tr>
</thead>
<tbody>
<tr>
<td>62042547</td>
<td>M(16.016,15.781,11.875) = 1.15222</td>
<td>$P_0$</td>
</tr>
<tr>
<td>61070205</td>
<td>M(15.469,11.641,11.406) = 1.14404</td>
<td>$P_1$</td>
</tr>
<tr>
<td>61034645</td>
<td>M(15.469,11.641,11.250) = 1.1309</td>
<td>$P_2$</td>
</tr>
<tr>
<td>24761257</td>
<td>M(12.422,14.219,8.984) = 1.12487</td>
<td>$P_3$</td>
</tr>
<tr>
<td>66105631</td>
<td>M(18.594,15.547,11.250) = 1.10726</td>
<td>$P_4$</td>
</tr>
<tr>
<td>24670347</td>
<td>M(12.578,14.219,8.984) = 1.10726</td>
<td>$P_5$</td>
</tr>
<tr>
<td>73113017</td>
<td>M(3.906,15.781,10.156) = 1.05001</td>
<td>$P_6$</td>
</tr>
<tr>
<td>62042547</td>
<td>M(16.016,15.781,11.875) = 1.07439</td>
<td>$P_0$</td>
</tr>
<tr>
<td>14204344</td>
<td>M(12.578,2.656,6.016) = 1.04675</td>
<td>$P_7$</td>
</tr>
<tr>
<td>66114730</td>
<td>M(18.594,15.547,16.016) = 1.04647</td>
<td>$P_3$</td>
</tr>
<tr>
<td>31107312</td>
<td>M(17.812,1.016,10.703) = 1.04467</td>
<td>$P_6$</td>
</tr>
<tr>
<td>74720664</td>
<td>M(1.719,14.297,18.125) = 1.03361</td>
<td>$P_7$</td>
</tr>
</tbody>
</table>

of attraction for an optimum and hence strengthening its exclusion.

C. Diversity among many optima
Equation 3 has an infinite number of optima. A picture of Equation 3 in two dimensions is given in Figure 3. The circular patterns are caused by the initial sine term of the equation. It divides space with a rectilinear grid with the “lines” of the grid being the points where $\sin(x_i) = 0$ for some $x_i$. Optima exist inside the grids and have varying height. The algorithm used in Section III-B for the eight optima function given in Equation 2 is used to optimize Equation 3 with the following modifications. The generators are still the vertices of a cube, but with $a = 70$, giving roughly 1000 optima inside the convex hull of the generators. The depth of representation is $r = 12$, and the mutation operator changes 6 characters. Both the original Sierpinski representation EA and the MOSS algorithm are used. The MOSS algorithm is run with an exclusion depth of 4, as before. The optima located are saved, and the number of times each optima is located is computed. The original algorithm found 356 optima, locating one in 13 separate runs. The MOSS algorithm found 969 optima and found no optima more than twice. The number of optima located a given number of times in 1000 trials for each algorithm is given in Table IV.

IV. CONCLUSIONS AND DISCUSSION
The Sierpinski representation is a novel representation for real optimization. Among its properties are the following.

- Highly controllable search with easily adjusted resolution. Resolution may be adjusted by fixing initial positions of the Sierpinski strings to shrink the search space, by extending the strings with neutral mutation to increase resolution, or by moving the generators to search a narrower or broader portion of the space. These ideas have yet to be tested.

- String-based encoding permitting easy sorting and identification of optima. This property permits the MOSS algorithm to function. The ability to use string algorithms to sort and classify optima in multi-dimensional real space may have applications well beyond the simple trick that permits the MOSS algorithm to locate many optima.

- The Sierpinski representation conducts provably well-spaced search of the convex hull of the generators if those generators form a hypercube or its stretched and sheared equivalents.

- Search with the Sierpinski representation takes place only inside the convex hull of the generators. This is both a limitation (although it can be overcome) and a potential
entry point for expert knowledge. It also permits a divide-and-conquer approach to searching a space in which the space is subdivided and the boundaries of the subdivisions are used as generators.

The parameter setting studies show that choosing the correct depth of representation impacts performance but is not critical - both the depth 6 and depth 12 representations performed adequately in the parameter setting study. The number of characters changed by the mutation operator is also an important parameter with 1 or 2 characters being the best number to change. Most of the runs using the Sierpinski representation were significantly faster than those using the baseline algorithm. A broader test on a larger number of functions and a more thorough parameter setting study are an early priority for additional research.

The success of the Sierpinski representation on the first test problem merits some discussion. The function has two optima and is constructed so that the lower optima is substantially easier to find because it basin of attraction, for a simple hill-climber, occupies more of the search space. The use of a fourth power curve on the bottom of the optima also means that values close to the value at the peak of this optima happen at a greater distance from the optima. In short, the inferior optima is designed to be a trap for a standard hill climber. Both EAs, the baseline and the Sierpinski, are able to avoid this trap by the use of a fairly large population size of 540 structures. We conjecture that the Sierpinski representation gains its advantage by climbing hills in a radically different manner from the standard algorithm. Even at mutation rates that come close to re-randomizing a gene (6 mutations with depth of representation 6) the Sierpinski representation performs on a par with the baseline EA. At lower mutation rates the Sierpinski representation significantly outperformed the standard one, though both find this test problem fairly easy, requiring only a few hundred mating events on average to find a solution.

A feature of the Sierpinski representation is that the impact on position within the search space and hence fitness is strongly weighted toward the prefix of the representation. This means that the convergence to an optima when using this method of search is domino convergence[6]. In a given run or the algorithm the population fixes the prefix first and then continues to search the tail of the strings only once the prefix becomes fixed. This feature is neither intrinsically good not bad, rather it means that this representation will interact with problems in a different manner from the standard ones. This feature is probably related to the algorithm’s ability to function at high mutation rates: only the first couple characters need survive. Recall that the mutated positions are chosen with replacement and so six mutations need no affect six distinct characters.

### A. Utility of the MOSS Algorithm

In the experiments on Equation 2, which has eight optima, the non-MOSS version of the Sierpinski representation found the global optimum 85% of the time, one of the best two optima 99.8% of the time, and never found the worst five optima at all. This suggests substantial efficiency in locating good optima. The MOSS version of the algorithm located all eight optima in its first 12 runs. If an optima was located more than once, this increased the degree to which it was subsequently excluded from search. When optimizing Equation 3 the MOSS algorithm located 969 optima in 1000 trials compared with 356 for the non-MOSS Sierpinski representation algorithm. The MOSS algorithm is highly effective at locating a diversity of optima. In difficult search problems, ones with very narrow optima, this could translate into better global optimization ability - so long as the radius of exclusion implied by the exclusion depth does not mask the entire basin of attraction for narrow optima.

The computational cost of the MOSS algorithm is low. Before the fitness of a string is evaluated, a tree-structured dictionary is checked to see if the string has the same prefix as a known optimum. If it does, it is awarded a fitness of zero (all the functions being optimized in this study are strictly positive). Such a tree-structured directory can be searched in time proportional to the depth of representation or the exclusion depth, whichever is shorter. If fitness evaluation is at all expensive, the cost of dictionary exclusion of this sort is nugatory or, since it prevents evaluation of some genes, potentially a time-savings. In other words the MOSS algorithm may be faster than the non-MOSS if individual points are expensive to evaluate. The Sierpinski representation does have the cost of Algorithm 2 added to each fitness evaluation. This costs a number of averagings equal to the depth of representation - not too costly, but not cheap enough to ignore.

<table>
<thead>
<tr>
<th>Times located</th>
<th>Number of Optima</th>
<th>MOSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>122</td>
<td>938</td>
</tr>
<tr>
<td>2</td>
<td>77</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>56</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Table IV**

Relative rate of location among the 1000 populations run optimizing Equation 3 for the original and MOSS algorithms.
V. NEXT STEPS

This study establishes that the Sierpinski algorithm is superior to a standard evolutionary algorithm using many of the same parameters at optimizing a mildly deceptive bimodal function. Speed comparisons on a broader selection of functions will be made in subsequent studies. During these studies the potential of the algorithm to refine it search by adding additional generators that enter the population via mutation can also be performed. If the position, in Euclidean space, of a current best gene is added as a generator then a copy of the entire current search space both centered on that generator and with a higher resolution is added to the search space. This should permit an efficient shift in the balance of exploration and exploitation in the evolutionary search.

A. Multi-objective Optimization

The ability of the MOSS algorithm to locate multiple optima makes it a natural tool for multi-objective optimization[2]. Once a point on the pareto-frontier has been located it can be stored and avoided. The algorithm can be used to search a pareto-frontier at a specified depth of representation by specifying the representation depth at which the MOSS algorithm excludes previously located points. Testing the MOSS algorithm on multi-objective problems is an early priority. A test suite such as that found in [3] is a natural choice.

B. The Impact of Sparse Representation

The experiments performed in this study were all performed in subsets of $\mathbb{R}^2$ and $\mathbb{R}^3$. Theorem 1 and its corollary require $2^n$ generators to guarantee complete search of a given subset of $\mathbb{R}^n$. If we are optimizing a twenty-parameter function this would require $2^{20} = 1048576$ characters in the alphabet, not a practical proposition. If fewer generators are used then a situation akin to the one in Figure 1 will hold: the search will have holes in it. To put it another way, the Sierpinski representation with fewer than $2^n$ generators samples but does not thoroughly search a space. In order to sample $n$-space only $n+1$ generators are required. Such a situation is called a sparse Sierpinski representation. A number of techniques may permit efficient functioning of a sparse representation.

- replace generators not used frequently with best-so-far optima. This will focus search near this best-so-far point. This may be a happy medium between density of coverage and excess alphabet size.
- Sample the space and select an acceptable number of generators placed according to principle components analysis of the fitness variation.
- Randomly place an acceptable number of generators in the region of interest. If the generators are placed differently in each run, then the aggregate coverage may be relatively complete.
- Hybridize the Sierpinski representation with a hill-climbing algorithm that operates directly on the real variables. The hill-climber will fill in the gaps in the Sierpinski representation.

Many other variations are possible and techniques for effective use of sparse Sierpinski representations are required if they are to be used for parameter optimization problems with more than 5-6 parameters.

C. Training Neural Nets

A nice place to examine sparse representations would be in the training of artificial neural nets. Standard examples such as $n$-parity or simple function approximation would be a natural environment in which to check the viability of the sparse Sierpinski representation. The Sierpinski representation can be used to document hilltops in the neural net training space and classify distinct neural net solutions. In addition, if the generators are competent nets trained by methods such as back-propagation, a hybrid search for nets that do not overgeneralize or which are robust may be made. Back propagation provides competent nets that serve as generators while a Sierpinski-based evolutionary algorithm permits evolutionary search on secondary criteria such as stability or good cross-validation behavior.

VI. ACKNOWLEDGMENTS

The first author would like to thank the Canadian Natural Sciences and Engineering Research Council as well as the University of Guelph’s Department of Mathematics and Statistics for its support of this research.

REFERENCES