The Challenge of Irrationality: Fractal Protein Recipes for PI

Jean Krohn
Department of Computer Science
University College London
Malet Place, London
J.Krohn@cs.ucl.ac.uk

Peter J Bentley
Department of Computer Science
University College London
Malet Place, London
P.Bentley@cs.ucl.ac.uk

Hooman Shayani
Department of Computer Science
University College London
Malet Place, London
H.Shayani@cs.ucl.ac.uk

ABSTRACT
Computational development traditionally focuses on the use of an iterative, generative mapping process from genotype to phenotype in order to obtain complex phenotypes which comprise regularity, repetition and module reuse. This work examines whether an evolutionary computational developmental algorithm is capable of producing a phenotype with no known pattern at all: the irrational number PI. The paper summarizes the fractal protein algorithm, provides a new analysis of how fractals are exploited by the developmental process, then presents experiments, results and analysis showing that evolution is capable of producing an approximate algorithm for PI that goes beyond the limits of precision of the data types used.

Categories and Subject Descriptors
F.4.2 [**Grammars and Other Rewriting Systems**]: Parallel rewriting systems (e.g., developmental systems).

General Terms
Algorithms, Design, Experimentation, Theory.

Keywords
Fractal Proteins, Computational Development, evolutionary computation, PI, irrational numbers, pattern.

1. INTRODUCTION
In nature there is no direct mapping from genotype to phenotype. Organisms grow and develop. Their genetic instructions operate like highly parallel programs that define how that growth should occur. The result is the complexity of life.

Researchers in the growing field of Computational Development argue that by adding a similar extra mapping from genotype to phenotype in an evolutionary algorithm, we can achieve similar advantages to those observed in nature. For example, because the genotype now corresponds to a generative program, our solutions become capable of fault-tolerance, self-repair and may develop into complex solutions that can become greater in scale by simply running the developmental program for longer [9-12,14,15].

But development is typically an iterative process involving the execution of genetic instructions repeatedly and the creation and reuse of building blocks or modules [16]. This is ideal if your solution is complex, containing regularities and repetitions. But what if your solution appears to be both complex and highly irregular? What if the solution is non-random, but contains no known pattern of any kind? Can evolution create a developmental algorithm that is capable of producing a phenotype with no patterns? Can a developmental approach generate the irrational number PI?

If the answer is yes, then we will have demonstrated yet another feature of development – the ability for evolution to exploit an iterative, pattern-rich process to create non-pattern, in the form of a very precise irrational sequence of digits. While number theory in mathematics may suggest this is possible, this work aims to investigate how such a challenge can be met.

The work described here continues an earlier investigation into the use of fractals as a computer representation of proteins. Earlier work has shown that fractal proteins are highly evolvable by a genetic algorithm [2][5], that specific patterns of activation in a fractal gene regulatory network (GRN) can be evolved [2][4], that evolved fractal GRNs naturally show fault-tolerance [5], and that they can perform computational tasks such as function regression and robot control [3]. This work exploits a complete reimplementation of the fractal-protein-based evolutionary developmental system by the first author and initially focuses on the challenge of irrationality: evolving and developing the irrational number PI as accurately as possible.

2. BACKGROUND
Early work in the area of computational development examined the evolution of embryogenies for specific target shapes, e.g. letters of the alphabet [17] or tessellating tiles [9]. Since then, researchers such as Hornby [12], Bongard [8], Gordon and Bentley [10] and Kumar and Bentley [16] demonstrated that various types of development can enable smaller genotypes to represent more complex phenotypes through the ability of development to discover modularities and repetition. Miller described experiments evolving developmental programs to create “French Flag” patterns [20]. They showed that development is able to regenerate these patterns, and that different patterns can be evolved in different environments. The research was the first of many similar attempts to evolve 2D target flag shapes.

But evidence suggests that when using implicit embryogenies (where repeated rules result in the emergence of the

GECCO'09, July 8–12, 2009, Montreal, Quebec, Canada.
Copyright 2009 ACM 978-1-60558-325-9/09/07...$5.00.

phenotype), at least some degree of phenotypic regularity is necessary for an advantage to be gained by the use of development [9,10]. As this work shows, this notion is not always valid. The irrational number PI is an excellent example. When regarded as a phenotype, it is a very difficult target for evolution, for it comprises an infinitely long sequence of precise digits containing no known (infinitely repeating) patterns. Should a direct mapping from genotype to phenotype be used with a simple genetic algorithm, clearly evolution to a certain limited precision would be possible, given sufficient time. But development provides the potential for creating not just the sequence, but an algorithm for generating that sequence. Thus, if evolution could find the right developmental algorithm, there would be no theoretical limit to the number of digits of PI that could be generated.

However, algorithms for generating PI are not simple. Bentley [1] discusses some of the history, summarized here. For example, Archimedes created one approach by creating polygons that fitted inside and outside a circle to calculate the value. Using polygons of 96 sides he calculated that PI lay between 22/7 and 233/71 – so he calculated PI to 1 decimal place. It took many more centuries before mathematicians could improve on this calculation. A German mathematician named Van Ceulen spent most of his life using Archimedes' approach, creating polygons with an almost unbelievable 4,611,686,018,427,387,904 sides in order to calculate PI to 35 decimal places. In the 17th Century, Wallis discovered a series that converged to PI:

$$2 / PI = (1\times3\times3\times5\times5\times7\times\ldots) / (2\times2\times4\times4\times6\times6\times\ldots)$$

And at around the same time Gregory discovered another sequence:

$$PI / 4 = 1 - 1/3 + 1/5 - 1/7 + \ldots$$

These sequences clearly prove that a regular pattern can produce the irregular number PI. However, both require tens of thousands of terms in the sequence before any real precision is reached, making this still a potentially very difficult challenge for evolution to duplicate. Gregory also discovered a sequence that requires fewer terms:

$$PI / 6 = (1/\sqrt{3})(1 - 1/(3\times3) + 1/(5\times3\times3) - 1/(7\times5\times3\times3) + \ldots$$

This only needs nine terms to achieve PI to a precision of 4 decimal places. Nevertheless, the complexity of this sequence implies that even an approximate developmental algorithm for generating PI may be a significant challenge for evolution.

This work uses the fractal protein model of computational development to tackle the problem, exploiting its known evolvability [5] and complex protein and gene interactions [2-4] to provide a rich set of computational tools for the task.

3. FRACTAL PROTEINS

Development is the set of processes that lead from egg to embryo to adult. Instead of using a gene for a parameter value as we do in standard EC (i.e., a gene for long legs), natural development uses genes to define proteins. If expressed, every gene generates a specific protein. This protein might activate or suppress other genes, might be used for signalling amongst other cells, or might modify the function of the cell it lies within. The result is an emergent "computer program" made from dynamically forming gene regulatory networks (GRNs) that control all cell growth, position and behaviour in a developing creature [23].

Table 1. Types of objects in the representation

fractal proteins	defined as subsets of the Mandelbrot set.
Environment	contains one or more fractal proteins (expressed from the environment gene(s)), and one or more *cells*.
Cell	contains a *genome* and *cytoplasm*, and has some *behaviours*.
Cytoplasm	contains one or more fractal proteins.
Genome	comprising *structural genes* and *regulatory genes*. In this work, the structural genes are divided into different types: *cell receptor genes, environment genes* and *behavioural genes*.
regulatory gene	comprising operator (or promoter) region and coding (or output) region.
cell receptor gene	a structural gene with a coding region which acts like a mask, permitting variable portions of the environmental proteins to enter the corresponding cell cytoplasm.
environment gene	a structural gene which determines which proteins (maternal factors) will be present in the environment of the cell(s).
behavioural gene	structural gene comprising operator and cellular behaviour region.

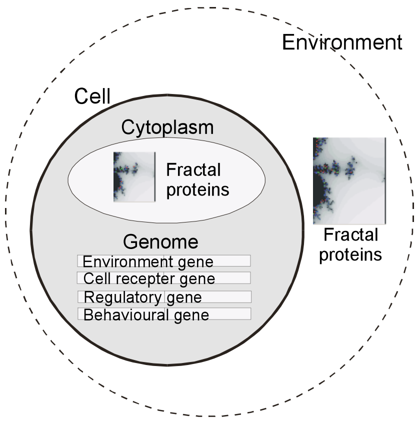


Figure 1. Representation using fractal proteins.

```
FRACTAL DEVELOPMENT

For every developmental time step:
  For every cell in the embryo:
      Express all environment genes and
      calculate shape of merged environment fractal proteins

      Express cell receptor genes as receptor fractal proteins
      and use each one to mask the merged environment proteins
      into the cell cytoplasm.

      If the merged contents of the cytoplasm match a promoter
      of a regulatory gene, express the coding region of the gene,
      adding the resultant fractal protein to the cytoplasm.

      If the merged contents of the cytoplasm match a promoter of a
      behavioural gene, use coding region of the gene to specify a
      cellular function.

      Update the concentration levels of all proteins in the cytoplasm.
      If the concentration level of a protein falls to zero, that protein
      does not exist.
```

Figure 2. The fractal development algorithm.

In this work, a biologically plausible model of gene regulatory networks is constructed through the use of genes that are expressed into *fractal proteins* – subsets of the Mandelbrot set that can interact and react according to their own fractal chemistry. Further motivations and discussions on fractal proteins are provided in [2-5]. Table 1 describes the object types in the representation; Figure 1 illustrates the representation. Figure 2 provides an overview of the algorithm used to develop a phenotype from a genotype. Note how most of the dynamics rely on the interaction of fractal proteins. Evolution is used to design genes that are expressed into fractal proteins with specific shapes, which result in developmental processes with specific dynamics.

3.1 Defining a Fractal Protein

In more detail, a fractal protein is a finite square subset of the Mandelbrot set, defined by three codons (*x,y,z*) that form the coding region of a gene in the genome of a cell. Each (*x, y, z*) triplet is expressed as a protein by calculating the square fractal subset with centre coordinates (*x,y*) and sides of length *z*, see fig. 3 for an example. In addition to shape, each fractal protein represents a certain *concentration* of protein (from 0 meaning "does not exist" to 200 meaning "saturated"), determined by protein production and diffusion rates.

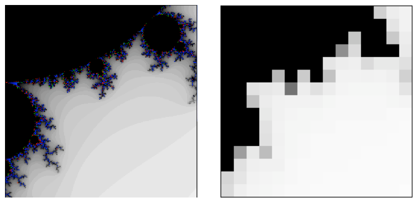


Figure 3. Example of a fractal protein defined by (*x*=0.132541887, *y*=0.698126164, *z*=0.468306528) Left: high resolution view. Right: actual sampling resolution.

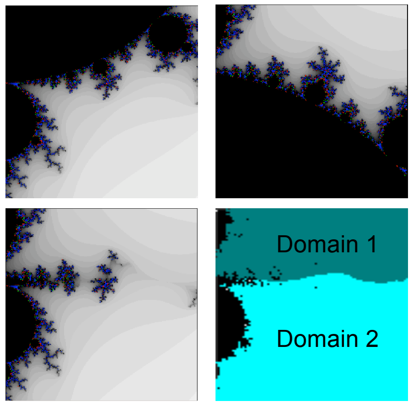


Fig. 4. Top: two fractal proteins; Bottom left: the resulting merged fractal protein combination; Bottom right: the two protein domains making up the merged protein combination, illustrating that the top-left protein forms the bottom two-thirds of the shape and the top-right protein forms the top third of the shape.

3.2 Fractal Chemistry

Cell cytoplasms and the environment usually contain more than one fractal protein. In an attempt to harness the complexity available from these fractals, multiple proteins are merged. The result is a product of their own "fractal chemistry" which naturally emerges through the fractal interactions.

Fractal proteins are merged (for each point sampled) by iterating through the fractal equation of all proteins in "parallel", and stopping as soon as the length of any is unbounded (i.e. greater than 2). Intuitively, this results in black regions being treated as though they are transparent, and paler regions "winning" over darker regions. See fig 4 for an example. Only the concentration values corresponding to the winning protein domains contribute to the overall concentration. Thus, the total concentration of two or more merged fractal proteins is the mean of the different protein concentrations in their merged product (e.g., in figure 4 the total concentration will be approximately one third of the concentration of the top-right protein plus two-thirds of the concentration of the top-left protein). If the value of more than one merged protein is identical at a sampled point, arbitration uses gene order. Concentrations slowly decrease over time to model diffusion. See table 1 and [2-5] for further details.

3.3 Genes

All genes contain 9 real-coded values:

xp	*yp*	*zp*	*Affinity threshold*	*Concentration threshold*	*x*	*y*	*z*	*type*

where (*xp, yp, zp, Affinity threshold, Concentration threshold*) defines the promoter (operator or precondition) for the gene and (*x,y,z*) defines the coding region of the gene. (*Affinity threshold* and *type* are stored as integers.) The *type* value defines which type of gene is being represented, and can be any combination of the following: *environment, receptor*, *behavioural*, or *regulatory*. This enables the type of genes to be set independently of their position in the genome, enabling variable-length genomes. It also enables genes to be multi-functional, i.e. a gene might be expressed both as an environmental protein and a behaviour.

When *Affinity threshold* is a positive value, one or more proteins must match the promoter shape defined by (*xp,yp,zp*) with a difference equal to or lower than *Affinity threshold* for the gene to be activated. When *Affinity threshold* is a negative value, one or more proteins must match the promoter shape defined by (*xp,yp,zp*) with a difference equal to or lower than |*Affinity threshold*| for the gene to be repressed (not activated).

To calculate whether a gene should be activated, all fractal proteins in the cell cytoplasm are merged (including the masked environmental proteins) and the combined fractal mixture is compared to the promoter region of the gene. Given the similarity matching score between cell cytoplasm fractals and gene promoter, the activation probability *Pa* of a gene is given by:

$$Pa = (1 + \tanh((m - At - C_t) / C_s)) / 2 \qquad \textbf{Equation 1}$$

where:
m is the matching score,
At is Affinity threshold (matching threshold from gene promoter)
C_t is a threshold constant (set to 0 in the experiments)
C_s is a sharpness constant (set to 20 in the experiments)

Every time step the new concentration of each protein is calculated. This is formed by summing two separate terms: the previous concentration level after diffusion (*Dc*) and the new concentration output by a gene (*Gc*). These two terms model the reduction in concentration of proteins over time, and the production of new proteins over time, respectively, where:

$Dc = Pc - Pc / C_p + 0.2$ **Equation 2**

Pc is protein concentration in previous time step,
C_p is a constant normally set to 5; the final addition of 0.2 ensures a minimum level of diffusion
and:

$Gc = Tc \times Cm$, **Equation 3**

Tc is the mean concentration seen at the promoter,
Cm is a concentration multiplier, where:

$Cm = \tanh((Tc - ct) / C_w) / C_i$ **Equation 4**

where: ct is the concentration threshold from the gene promoter
C_w is a constant (set to 30 for these experiments)
C_i is a constant (set to 2 for these experiments)

The full details of this process are beyond the scope of this paper, interested readers should consult [2-5].

Behavioural Gene. A behavioural gene is activated when other protein(s) in the cytoplasm match its promoter region (using the *affinity threshold*). For this application, a gradual activation between not activated and activated was required, using the *x* value of the coding region (*x,y,z*) triplet as a *fate* value to define a function, calculated as follows:
If the gene is being activated with a negative *Affinity threshold*,

out = *out* - (*totalconcentration* - *concentrationthreshold*) * *fate*

If the gene is being activated with a positive *Affinity threshold*,

out = out + (*totalconcentration* - *concentrationthreshold*) * *fate*

Note how the total concentration of proteins seen on the promoter is offset against the *Concentration Threshold* gene and scaled by the *fate* gene (*x* value of the coding region), allowing evolution to adjust the range of values seen on the output, and used to specify behaviours.

3.4 Development and Evolution

As was illustrated in figures 1 and 2, an individual begins life as a single cell in a given environment. To develop the individual from this zygote into the final phenotype, fractal proteins are iteratively calculated and matched against all genes of the genome. Should any genes be activated, the result of their activation (be it a new protein, receptor or cellular behaviour) is generated at the end of the current cycle. Development continues for *d* cycles, where *d* is dependent on the problem. Note that if one of the cellular behaviours includes the creation of new cells, then development will iterate through all genes of the genome in all cells.

All genes are evolved. The genetic algorithm used in this work has been used extensively elsewhere for other applications (including GADES [6]). A dual population structure is employed, where child solutions are maintained and evaluated, and then inserted into a larger adult population, replacing the least fit. The fittest *n* are randomly picked as parents from the adult population. Typically the child population size is set to 80% of the adult size and *n* = 40%. (For further details of this GA, refer to [6].) Because real coding was used, duplication and creep mutation is used, see [2] for complete details. Crossover is always applied; all mutations occur with probability 0.01 per gene.

3.5 How Fractal Development Works

In the merged protein shape, each sampled pixel value is the maximum corresponding pixel value from all proteins with non-zero concentrations. This makes the merged protein shape a patchwork of complex regions each belonging to one of the proteins present in the cytoplasm. We term the set of pixels in the merged protein originating from one protein as the domain of that protein. Figure 4 (bottom right) shows an example of two protein domains in the merged protein. If concentration of a protein drops to zero during development, that protein does not exist and so cannot have a domain in the merged result; instead other proteins may fill the region with their domains. This results in changes in the shape of those protein domains. This is analogous to the protein-protein interactions in biology resulting in proteins shifting their shapes.

The value of different pixels in the merged protein at each development time step can together represent a single point coordinate in a multidimensional state space, each dimension being the value of one pixel. We shall refer to this state space as the MPS space (merged protein state space). The pixel values of the promoter and the absolute value of the *Affinity threshold* collectively describe a convex subspace in the MPS space (gene expression subspace), specifying when this gene can be expressed. This creates a slightly different GRN (gene regulatory network) for each combination of present proteins. The expression of each gene is affected only by the pixel values of those protein domains that lie under the domain of the gene promoter. The sign of the *Affinity threshold* determines if this gene is expressed or repressed when the current MPS dwells inside this subspace. The absolute value of the *Affinity threshold* specifies the size of this subspace. The hyperbolic tangent function (in Equation 1 for *Pa* described in section 3.3) creates a smooth transition for probability of gene expression at the surface of this subspace. This can improve the evolvability of the GRNs by smoothing the fitness landscape. The concentrations of individual unmerged proteins at each developmental time step can together represent a single point coordinate in a multidimensional state space, each dimension being the non-zero concentration value of one protein. We shall refer to this state space as the PCS space (protein concentration state space). When a gene is expressed, the concentration of the protein encoded in the gene is increased (or decreased) by a multiplicative sigmoid function (Equations 3 and 4 for *Gc* and *Cm* in section 3.3) of a linear combination of the concentration of those proteins with their domains covered by the gene promoter domain.

The fractal development algorithm can also be viewed from the perspective of pattern recognition, where the cell receptor gene performs input feature selection by masking some of the environment proteins. The rest of the GRN can be seen as a reservoir (Reservoir Computing [22]) or a Liquid State Machine [18]. From this viewpoint, genes work as leaky integrating nodes with a multiplicative sigmoid transfer function (as described in section 3.3), interacting through protein concentrations in a recurrent network. The areas of those protein domains that lie under a gene promoter domain define the input weights for that node (gene), and the *Concentration threshold* works as a bias. The behavioural genes work as the readout map (in LSM and RC [18,22]) translating the current multidimensional PCS into outputs. Even randomly generated reservoirs can be effectively used for pattern recognition and chaotic time-series prediction [13]. However, recent research [22] shows that bio-plausible features such as hierarchy and modularity in the reservoir network architecture can increase the performance and robustness of the reservoirs. Statistical studies also reveal such properties in biological GRNs [7]. Therefore it is quite likely that, using fractal protein domains, this system is able to evolve the suitable network structures for a given problem. Existence of inactive genes and complete (or partial) dominance of one protein domain on other protein domains result in neutral networks in the fitness landscape - another of the reasons for the evolvability of this system observed in [5]. Neutral mutations can drift the expression subspace of inactive genes. The randomness at the edge of these gene expression subspaces can give evolution some clues about the promising inactive genes that should be turned on to smoothly evolve a GRN into a fitter GRN.

4. EXPERIMENTS

Using the fractal developmental system, we aim to obtain a system that produces PI with an increasing precision. We examine two different ways to approximate PI using fractal proteins. First we aim to produce the binary representation of PI; second we aim to approximate its real value.

4.1 Experiment 1 – Binary Approximation

In the first experiment, the output of the system is the activation state of the first behavioural gene in the genome, i.e. we require one gene to switch on and off according to the pattern of PI written in base 2. The other outputs of the behavioural genes are ignored. If a genome does not contain a behavioural gene, which is possible as evolution is free to add/remove genes or change their types, the output of the system will be 0. It is worth noting that the binary representation of PI also contains no pattern:

1100 1001 0000 1111 1101 1010 1010 0010 …

An incremental fitness function is used. Initially, the system is only exposed to the first bit of PI. The system is then only exposed to a new bit of PI if it matches all of the previous bits correctly. The overall fitness value is a weighted sum of correct bits. A decreasing weight is given to each bit of the pattern, so that a pattern which only has the first bit correct is fitter than a pattern with the first bit incorrect and every other correct, and so on recursively. Unlike [2] and [4], the number of edges in the pattern obtained is not used in the fitness function. This is done so that the system would evolve the most accurate approximation of PI possible, and not just try to match a pattern.

Note that this is a more difficult task compared to evolving a fixed size pattern, as it has to evolve to fit a moving goal (as provided by the incremental fitness function), and may have to drastically change its developmental processes as it progresses through the bits of PI. Each individual develops for 32 iterations, so the fittest possible individual would achieve 32 correct bits of PI.

The size of the genetic algorithm total population is 100 individuals. The individuals have a lifespan of 10 generations. 40 of the best in the population are used to generate 80 children which replace the worst, each generation. The GA evolved for 1000 generations. The experiment was run 100 times.

4.2 Experiment 2 – Value Approximation

In the second experiment we evolve a fractal gene regulatory network in an attempt to obtain an algorithm for the generation of PI. Ideally the developmental PI algorithm should provide an improving approximation to the value of PI, the more developmental iterations it is run. Unless specified otherwise, the settings are the same as specified for experiment 1.

For each developmental iteration, the output of the first behavioural gene is used as a scaling factor; the mean of the output of the other behavioural genes is divided by the product of this and the previous scaling factors, and summed to the total output. If a scaling factor is equal to zero, it will be ignored. If an individual's genome contains zero or one behavioural gene, the total output will be zero. The total value output is thus:

$$\sum_{i=1}^{I} \frac{\frac{\sum_{n=2}^{N} B_{n,i}}{N-1}}{\prod_{t=1}^{i} B_{1,i}}$$

Where:

- I is the number of developmental iterations.
- N is the number of behavioural genes.
- $B_{n,i}$ is the output of the n^{th} Behavioural gene in the genome, at developmental iteration i.

The system is evaluated for 32 developmental iterations. The fitness function used is simply the absolute difference between the total output and PI. Note that the system currently uses double types (which uses 52 bits for fractional part) in C++ to perform mathematical operations and store the values of PI, so precision is limited to 15 decimal places.

5. RESULTS

5.1 Results 1 – Binary Approximation

As can be seen in figure 5 (top) and table 2, the system was able to evolve a pattern of binary PI of up to 26 bits, with a decimal equivalent of 3.1415926.

This approach demonstrates the ability of fractal proteins to evolve a solution to fit a moving target. Despite the target containing no pattern, this approach achieves a respectable accuracy of 7 decimal places.

5.2 Results 2 – Value Approximation

Figure 5 (bottom) and table 2 provide the results for the second experiment. Out of a hundred runs, the system managed to obtain the exact double value of PI four times, i.e. the value of PI was correctly evolved to 15 decimal places. It seems likely that if a higher precision had been used, an even better approximation would have been reached. After converging to as good an approximation of PI as possible, these developmental solutions kept producing that same value with the same internal developmental patterns even after being run for up to 500 developmental iterations, so it is possible that these systems would keep converging towards PI if they were able to run with a higher precision. (The current implementation uses C++ double types for its processing.)

Remarkably, these solutions achieved this result through varied means. In two of the runs, the system converged towards PI using a repeating pattern at increasingly smaller scales (see Figure 6), each time over-correcting its difference with PI, but always getting closer. In another run, it started lower, but slowly increased its output until it reached PI, never going above it.

Table 2 provides a summary of the median and best results obtained in both experiments. Figure 5 shows the percentage of runs that obtained each degree of precision for both experiments.

Table 2. The median and best results for both experiments.

	Median	**Best**
Experiment 1	3.1411	3.1415926
Experiment 2	3.141592658	3.141592653589793

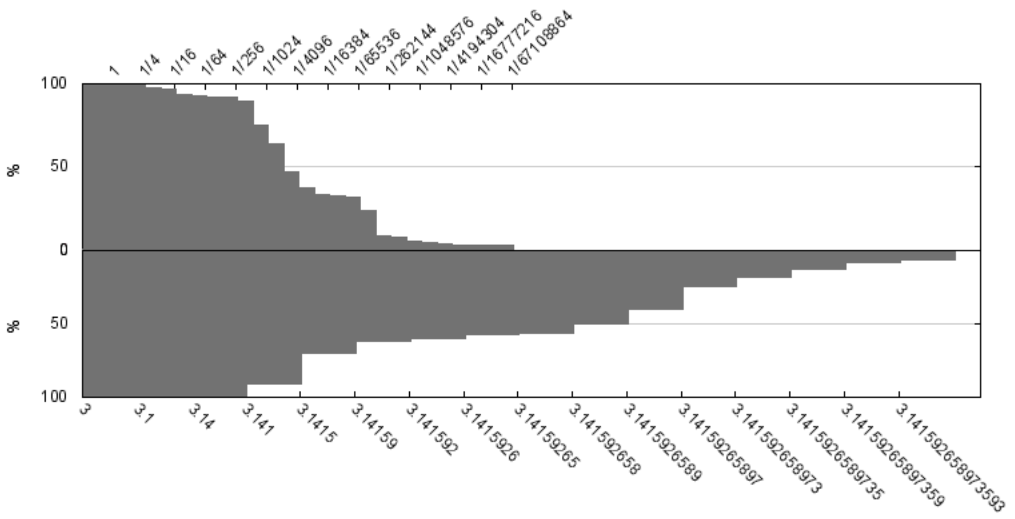


Figure 5 For each degree of precision, the percentage of runs that have reached it. Top: Results for Experiment 1. Bottom: Results for Experiment 2. The binary scale above corresponds to the decimal scale below in terms of accuracy.

a)

$$\text{output} = \sum_{i=1}^{I} \frac{\frac{\sum_{n=2}^{N} B_{n,i}}{N-1}}{\prod_{t=1}^{i} B_{1,i}}$$

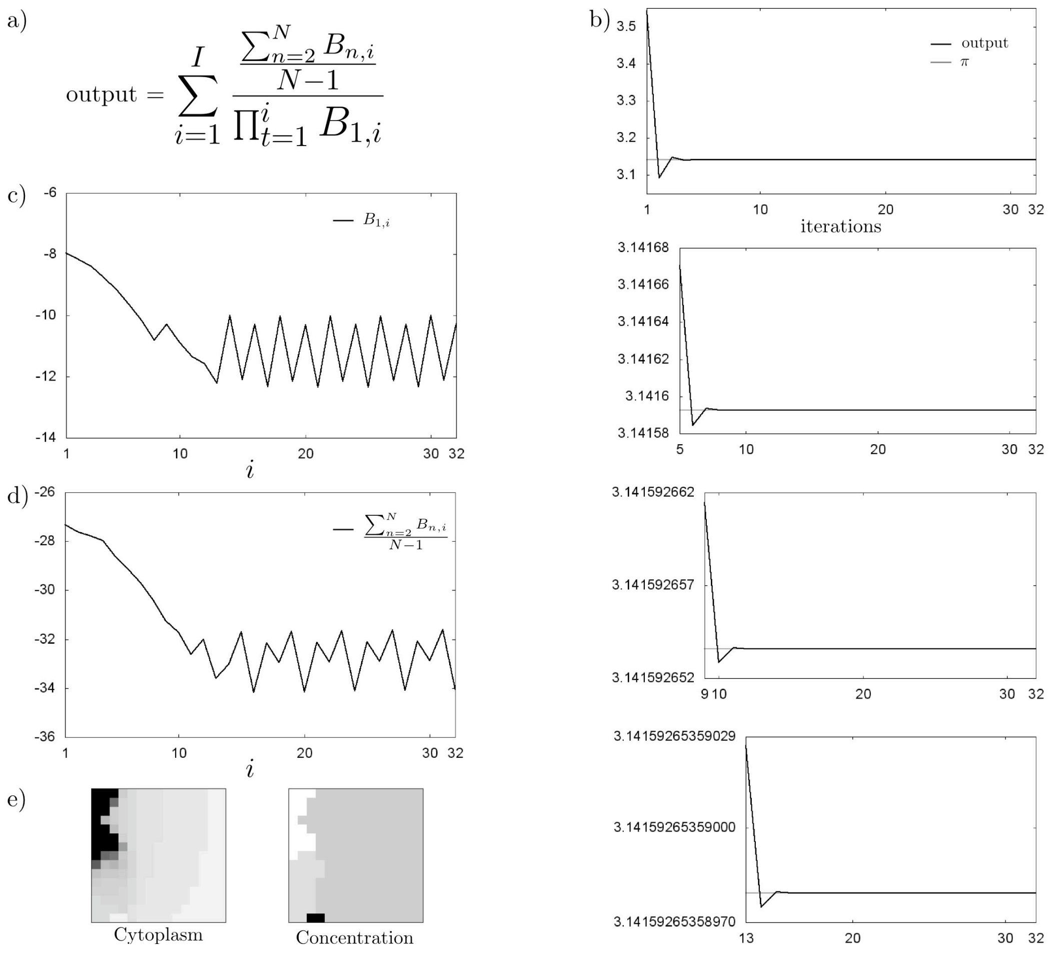
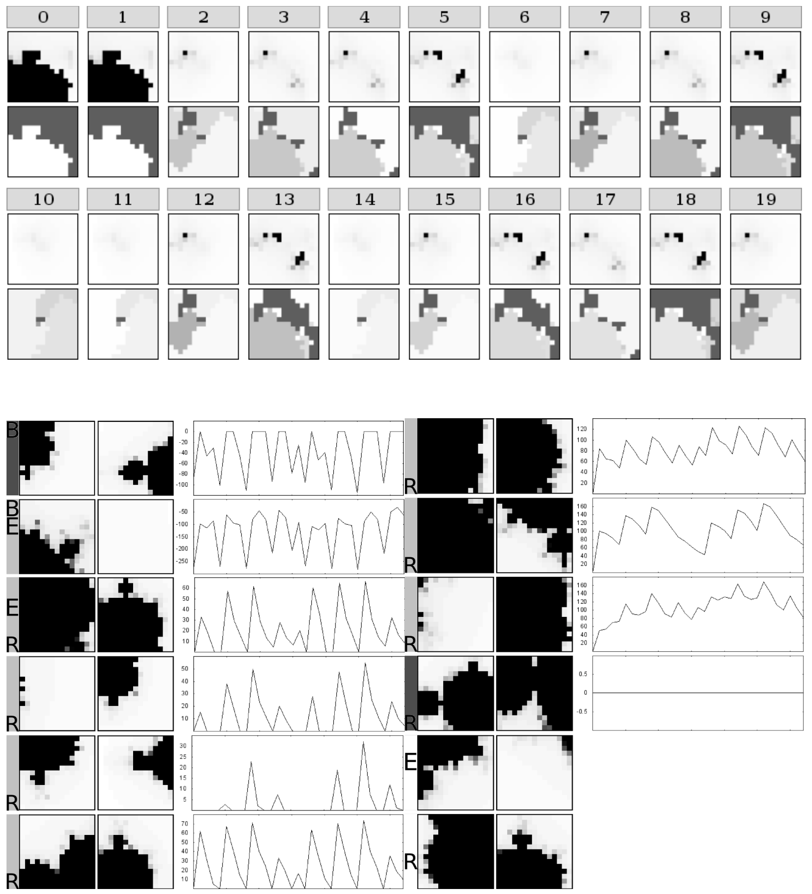
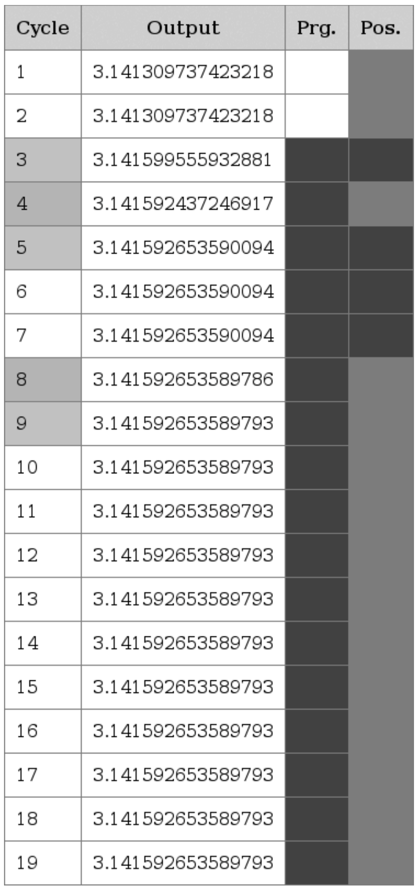


Figure 6 (a) The mathematical expression of the system's output (see section 4.2). (b) The output of a system producing PI to the maximum precision allowed by the double type; The same pattern is repeated at increasingly smaller scales, as development occurs. (c) The output of the first behavioural gene, the inverse of which is used as a scaling factor in (a). (d) The mean of the output of the other behavioural genes, which is the main component in (a). (e) An example of the cell's cytoplasm, with its associated concentration.





Cycle	Output	Prg.	Pos.
1	3.141309737423218		▨
2	3.141309737423218		▨
3	3.141599555932881	■	■
4	3.141592437246917	■	▨
5	3.141592653590094	■	■
6	3.141592653590094	■	■
7	3.141592653590094	■	■
8	3.141592653589786	■	▨
9	3.141592653589793	■	▨
10	3.141592653589793	■	▨
11	3.141592653589793	■	▨
12	3.141592653589793	■	▨
13	3.141592653589793	■	▨
14	3.141592653589793	■	▨
15	3.141592653589793	■	▨
16	3.141592653589793	■	▨
17	3.141592653589793	■	▨
18	3.141592653589793	■	▨
19	3.141592653589793	■	▨

Figure 7 The fractal protein developmental system running, illustrating another solution that achieved PI to 15 decimal places in only 9 iterations. The top-left panel shows the state of the cytoplasm for each developmental iteration. The merged proteins present in the cytoplasm is shown above, with the corresponding concentration at each point shown below; white indicates the absence of any protein and black indicates saturation. Iteration 0 corresponds to the state of the cytoplasm before any development, at which point it is a function of only the environmental and receptor genes. Several overlapping patterns can be observed in both the merging and the concentration. The bottom-left panel shows the individual genes and the fractal shapes of their promotor and coding regions, with concentrations of their output proteins over time (B = behavioural, middle R = receptor, bottom R = regulatory, E= environmental). To the right is shown the approximation produced by the system after each developmental iteration (or cycle). Dark grey for "Prg." indicates the current approximation is better. Dark grey for "Pos." indicates the approximation is above; light grey indicates the approximation is below the true value of PI (the precision of the code is exceeded by iteration 9 so the indication is inaccurate after that point).

ANALYSIS

Figure 7 shows the internal workings of the fractal developmental system for another of the best solutions evolved. It should be clear by examining the proteins and concentrations in the cytoplasm and the activations of individual genes over time that the system produces an internally repeating pattern (albeit sometimes very complex). This is typical of all the solutions evolved, and is typical of most implicit developmental systems simply because they involve repeated execution of generative or developmental rules [9]. It is not possible for the GA to evolve a constant that directly corresponds to PI (e.g. it cannot evolve an initial concentration value matching PI, for environmental proteins by default begin at the saturation value of 200). Instead, the GA has evolved a series of protein fractal shapes and threshold values that interact to form a relatively good approximation of the value, which is then iteratively improved by the gene-regulatory network as it continues to run. Again, this appears to be the typical solution chosen by evolution – whatever the initial starting value is, this value is incrementally improved, sometimes by a pattern that oscillates above and below the true value of PI, and sometimes by a pattern that creeps ever-closer to the value. These approaches to the generation of PI are surprisingly similar to the summed sequences described in section 2, which also incrementally improve their approximation to PI through the summing of successive terms of patterns. It is unlikely that any of the fractal developmental programs found here would produce a perfect value of PI (given that evaluation was limited to a fixed 32 developmental iterations and a fitness evaluation with a limited precision of 15 decimal places). Further work was subsequently performed using an extended precision for the output (but not for the internal fractal proteins). The best results were found to approximate PI correctly to 23 decimal places.

6. CONCLUSIONS

PI is one of our best examples of an infinite, non-random sequence of digits that contain no infinitely repeating pattern. Implicit developmental algorithms inherently and naturally produce patterns. Thus, the poor results obtained in experiment 1 were anticipated, when trying to evolve a fractal GRN with the non-pattern of PI. In experiment 2, we used a more appropriate mapping stage, permitting development to produce its patterns internally and then use those patterns to generate the value of PI incrementally as a summed series. The aim here was to encourage the evolution of an algorithm to keep giving better approximations of PI even when run beyond the number of iterations used during evolution.

It took mathematicians centuries to formulate algorithms to calculate PI. Today we know of several regular sequences that converge to PI, which illustrate that it is theoretically possible for a regular pattern to produce this famous example of non-pattern. These sequences are long and often complex. It was therefore unexpected and gratifying that in experiment 2, evolution was able to create approximate developmental algorithms that produced PI to the limits of the precision of the double data type. Further work showed that fractal development was able to go beyond the precision of the data type and produce PI to 23 decimal places.

However, it is clear that evolutionary development does not naturally produce phenotypic non-pattern. The relative success of this work was achieved by altering the task from the creation of an algorithm to generate individual digits into the production of an approximation algorithm for a specific numerical value. In our mathematics that value is written as a series of digits with no infinitely repeating pattern. In biology (and fractal development), which does not form solutions with basic mathematical operators or constants and does not “care” how numbers are written, the task of approximating PI is no different from the task of approximating any other value. When using protein concentrations in biology, only convergence to concentration levels and patterns of concentrations are possible. Thus very good approximations to values are feasible, but the creation of perfect values are unlikely. Biology is oblivious to PI, for it only exists in our mathematics; perfect circles are impossible. It is thus unlikely that a process that cannot directly exploit mathematical operators will be able to find a perfect algorithm for creating the digits of PI directly. The challenge of irrationality may be too great for biologically plausible evolutionary development.

7. REFERENCES

[1] Bentley, P. J. (2008) The Book of Numbers: The Secrets of Numbers and How They Created Our World. Cassell Illustrated. ISBN 1 84403 396 1. (Hardback, February 2008)

[2] Bentley, P. J. Fractal Proteins. In Genetic Programming and Evolvable Machines Journal. 2004.

[3] Bentley, P. J. Controlling Robots with Fractal Gene Regulatory Networks. Chapter in de Castro, L. and von Zuben, F. (editors) Recent Developments in Biologically Inspired Computing. Idea Group Inc, 2004.

[4] Bentley, P. J. Evolving Fractal Proteins. In Proc. of ICES ’03, the 5th International Conference on Evolvable Systems: From Biology to Hardware. 2003.

[5] Bentley, P. J. Evolving Beyond Perfection: An Investigation of the Effects of Long-Term Evolution on Fractal Gene Regulatory Networks. In Proc of *Information Processing in Cells and Tissues* (IPCAT). 2003.

[6] Bentley, P. J. From Coffee Tables to Hospitals: Generic Evolutionary Design. Chapter 18 in Bentley, P. J. (Ed) Evolutionary Design by Computers. Morgan Kaufmann Pub. San Francisco, 1999. pp. 405-423.

[7] Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., Hwang, D.: Complex networks: Structure and dynamics. Physics Reports 424(4–5) (2006) 175–308

[8] Bongard, J. C. Evolving Modular Genetic Regulatory Networks. In *Proc. of 2002 Congress on Evolutionary Computation*, IEEE Press, 2002. pp. 1872-1877.

[9] Bentley, P. J. and Kumar, S. Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem. Genetic and Evolutionary Computation Conference (GECCO '99), July 14-17, 1999, Orlando, Florida USA, 1999. pp.35-43. RN/99/2.

[10] Gordon, T. and Bentley, P. J. (2005) Development Brings Scalability to Hardware Evolution. In Proc. of 005 NASA/DoD Conference on Evolvable Hardware.

[11] P. C. Haddow, G. Tufte, and P. van Remortel. Shrinking the Genotype: L-Systems for EHW. Proc. 4th Int. Conf. on Evolvable Systems: From Biology to Hardware. 2001.

[12] Hornby, G. S. Generative Representations for Evolutionary Design Automation. Brandeis University, Dept. of Computer Science, Ph.D. Dissertation. 2003.

[13] Jaeger, H. & Haas, H. Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication Science, 2004, 304, 78-80

[14] A.H. Jackson, A.M. Tyrrell Implementing Asynchronous Embryonic Circuits using AARDVArc. In *Proc of NASA/DoD Conference on Evolvable Hardware* (EH-2002), IEEE Computing Society, Virginia, 2002. pp. 231-240.

[15] N. Jakobi. Harnessing Morphogenesis. *Int Conf Information Processing in Cells and Tissues*, Liverpool. 1995.

[16] S. Kumar and P. J. Bentley. Computational Embryology: Past, Present and Future. Invited chapter in Ghosh and Tsutsui (Eds) Theory and Application of Evolutionary Computation: Recent Trends. Springer Verlag (UK). 2003.

[17] Kumar, S. and Bentley, P. J. The ABCs of Evolutionary Design: Investigating the Evolvability of Embryogenies for Morphogenesis. A late-breaking paper in Genetic and Evolutionary Computation Conference (GECCO '99), July 14-17, 1999, Orlando, Florida USA, 1999. pp. 164-170

[18] Maass; Natschlager & Markram Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations NEURCOMP: Neural Computation, 2002, 14, 2531-2560

[19] Mandelbrot, B. *The Fractal Geometry of Nature*. W.H. Freeman & Company. 1982.

[20] Miller, J. and Banzhaf, W. Evolving the Program for a Cell: From French Flags to Boolean Circuits. Invited chapter in Kumar, S. and Bentley, P. J. (Eds) *On Growth, Form and Computers*. Academic Press, 2003.

[21] Quick, T. Evolving Embodied Genetic Regulatory Network-Driven Control Systems. Proc. of ECAL 2003.pp. 266-277.

[22] Schrauwen, B.; Verstraeten, D. & Van Campenhout, J.An overview of reservoir computing: theory, applications and implementations. Proceedings of the 15th European Symposium on Artificial Neural Networks, 2007, 471–482

[23] Lewis Wolpert, Rosa Beddington, Thomas Jessell, Peter Lawrence, Elliot Meyerowitz, Jim Smith. Principles of Development, 2nd Ed. 2001. Oxford University Press.