Solving Iterated Functions Using Genetic Programming

Michael D. Schmidt
Computational Synthesis Lab
Cornell University
Ithaca, NY 14853
mds47@cornell.edu

Hod Lipson
Computational Synthesis Lab
Cornell University
Ithaca, NY 14853
hod.lipson@cornell.edu

ABSTRACT
An iterated function $f(x)$ is a function that when composed with itself, produces a given expression $f(f(x))=g(x)$. Iterated functions are essential constructs in fractal theory and dynamical systems, but few analysis techniques exist for solving them analytically. Here we propose using genetic programming to find analytical solutions to iterated functions of arbitrary form. We demonstrate this technique on the notoriously hard iterated function problem of finding $f(x)$ such that $f(f(x))=x^2-2$. While some analytical techniques have been developed to find a specific solution to problems of this form, we show that it can be readily solved using genetic programming without recourse to deep mathematical insight. We find a previously unknown solution to this problem, suggesting that genetic programming may be an essential tool for finding solutions to arbitrary iterated functions.

Categories and Subject Descriptors
J.2 [**Physical Sciences and Engineering**]: Mathematics and statistics.

General Terms
Algorithms

Keywords
Iterated Functions, Symbolic Regression

1. INTRODUCTION
Solving an iterated function is a type of mathematical problem where the analytical form of a function $f(x)$ is not known, but its composition with itself is known. For example, what is $f(x)$ such that $f(f(x)) = g(x)$, where $g(x)$ is a given function.

Iterated functions appear in various fields such as fractal theory, computer science, dynamical systems, and maps. Despite their prevalence, they are notoriously difficult to solve, and few mathematical tools exist to analyze them. A few methods have been developed to analyze special cases. In the general case, solving these problems has involved deep mathematical insight and experimentation with different substitutions and reformulations [1] for each class of iterated function problem.

In particular, a challenging iterated function problem has circulated as a test of one's intelligence particularly among physicists [2] and in math competitions [3]. This problem asks to find an analytical function $f(x)$ such that $f(f(x)) = x^2 - 2$. In fact, those who first solved this problem are still celebrated within various communities. Renowned physicist Michael Fisher is rumored to have solved the puzzle within five minutes [2]; however, few have matched this feat.

The problem is enticing because of its apparent simplicity. Similar problems such as $f(f(x)) = x^2$, or $f(f(x)) = x^4 + b$ are straightforward (see Table 1). The fact that the slight modification from these easier functions makes the problem much more challenging highlights the difficulty in solving iterated function problems.

Table 1. A few example iterated functions problems.

Iterated Function	Solution
$f(f(x)) = x$	$f(x) = x$
$f(f(x)) = x - 2$	$f(x) = x - 1$
$f(f(x)) = x^4$	$f(x) = x^2$
$f(f(x)) = x^2 - 2$	$f(x) = ?$

In this paper, we propose using genetic programming to identify and solve such iterated functions. We adapt the symbolic regression algorithm to search for equations that iterate to the correct map and solve the $f(f(x)) = x^2 - 2$ problem.

The key benefit of using symbolic regression to solve this type of problem is that it does not require deep mathematical insight into the problem and it is free to find all solution forms to the iterated function. Current solutions to this problem require recognizing relations of Chebyshev polynomials or substituting special functional forms. These methods also make assumptions which lead to specific solutions, but there may and often do exist other valid or more general solutions. We show genetic programming can search for all solution types without assuming certain forms and find new solutions faster than even the best human problem solvers.

In the remaining sections, we provide a background on iterated functions and their known solution methods. We then detail our genetic programming approach and results, including a new solution to the notorious iterated function problem, before concluding with final remarks.

2. BACKGROUND
Iterated functions are known to present difficult mathematical challenges. Here, we overview various applications of iterated functions and known techniques for handling them.

GECCO'09, July 8–12, 2009, Montréal, Québec, Canada.
ACM 978-1-60558-505-5/09/07.

2.1 Iterated Functions

An iterated function is a mathematical function that is composed with itself one or more times (the output of a function is fed back into the same function one or more times). For iterated function problems, we are given the output of the function after iteration, and we are attempting to find the function that produces this output. For example:

$$f^n(x) = g(x)$$

where $g(x)$ is given and the notation $f^n(x)$ denotes that the function $f(x)$ is iterated n times, what is the function $f(x)$? Problems such as this arise in several fields from fractals, to computer science, and dynamical systems.

A fractal is produced by a system of one or more iterated functions – which may be graphical or algebraic functions. The inverse problem in iterated function systems is the problem of identifying the procedure that produces a fractal. For example, given the result of a simple fractal, what is the iterated function that produces that fractal? Evolutionary algorithms have been applied to the graphical version of problem [4]. In our case, we are looking at a solving a particularly challenging algebraic iterated function problem.

Iterated functions also arise in computer science. Approximating the iterated function has been applied to image compression [5], where a given image is approximated by finding a simple function that can reproduce it when iterated. Iterated functions also arise in lambda calculus and functional programming, where one is attempting to find a recursive function to compute a desired result.

In dynamical systems, iterated functions arise in finite difference equations and 1D maps [1]. A dynamical system can be modeled by an iterated equation. The iterated function problem arises here when the behavior of the dynamical system is known, but the difference or map function is unknown.

2.2 Analytical Solution Methods

In this section we overview a few of the basic mathematical techniques that have been deduced to solve certain families of iterated functions.

The most basic approach to solving an iterated function is to assume the function has some given structure, such as a polynomial structure. For example, given:

$$f(f(x)) = x^4 - 2$$

one might suspect that the function f(x) is also a polynomial of lesser degree. For example, assume $f(x)$ takes the following form:

$$f(x) = a\,x^2 + b\,x + c$$

Iterating this form yields:

$$f(f(x)) = a\,(a\,x^2 + b\,x + c)^2 + b\,(a\,x^2 + b\,x + c) + c$$

$$= a^3x^4+2a^2bx^3+(2a^2c+ab^2+ab)x^2+(2abc+b^2)x+ac^2+bc+c$$

Next, we solve for a, b, and c, by equating the coefficients to the known iterated function coefficients. We solve the set of equations $a^3 = 1$, $2a^2b = 0$, $2a^2c+ab^2+ab = 0$, $2abc+b^2 = 0$, and $ac^2+bc+c = -2$ for a, b and c of our assumed $f(x)$ polynomial.

This approach breaks down however whenever the function is not polynomial. In fact, some iterated functions produce polynomials, but cannot be solved as polynomials. For example, given:

$$f(f(x)) = x^2 - 2$$

solving $f(x)$ for polynomial coefficients fails. Yet, this problem does have a solution.

This specific example is a long standing problem in mathematics. Recently, different methods have been developed to handle this particular case [1, 2]. One clever approach is to substitute in the following functional form to rewrite $f(f(x))$:

$$f(x) = g(a\,g^{-1}(x)),$$

$$f(f(x)) = g(a^2\,g^{-1}(x)),$$

$$g(a^2\,g^{-1}(x)) = x^2 - 2,$$

where a is a parameter constant, and $g(x)$ is some other function that we may infer by inspection. For example, one may think to set $a^2 = 2$ and substitute $t = g^{-1}(x)$ to write:

$$g(2t) = x^2 - 2 = g(t)^2 - 2$$

which looks remarkably as a double angle formula, solved as:

$$x = g(t) = 2\cos(t),$$

$$x = g(g^{-1}(x)) = 2\cos(g^{-1}(x))$$

Next, we can solve for $f(x)$:

$$f(x) = 2\cos(\sqrt{2}\cos^{-1}(x/2))$$

A second method involves the application of Chebyshev polynomials. One may recognize the following identities of Chebyshev polynomials:

$$T_n(\cos(t)) = \cos(n\,t), \text{ and}$$

$$T_m(T_n(x)) = T_{m\cdot n}(x)$$

By inspection one could then rewrite:

$$f(f(2\cos(t/2))) = 2*\cos(n\,t/2),$$

$$f(f(2\cos(t/2)) = 2\cos(\sqrt{2}\;t/2),$$

$$f(x) = 2\cos(\sqrt{2}\cos^{-1}(x/2))$$

While these solutions are remarkable deductions, they also make assumptions on the form of $f(x)$ and cannot be applied to all iterated function problems. Below, we show that the problem can be solved without deep mathematical insight or assuming a particular form of the $f(x)$ solution.

3. OUR METHOD

We adapted the symbolic regression algorithm to search for solutions to the iterated function problem. Here we overview the symbolic regression algorithm, the fitness function for iterated functions, and how we verify the solutions found.

3.1 Symbolic Regression

Symbolic regression [6] is a type of genetic program for searching the space of expressions computationally by minimizing various error metrics. Both the parameters and the form of the equation are subject to search. In symbolic regression, many initially random symbolic equations compete to model experimental data in the most parsimonious way. It forms new equations by recombining previous equations and probabilistically varying their sub-expressions. The algorithm retains equations that model the experimental data better than others while abandoning unpromising solutions. After an equation reaches a desired level

of accuracy, the algorithm terminates, returning its most parsimonious equation that is most likely to correspond to the intrinsic mechanisms of the observed system.

In symbolic regression, the genotype or encoding represents symbolic expressions in computer memory. Often, the genotype is a binary tree of algebraic operations with numerical constants and symbolic variables at its leaves [7, 8], for example, a binary parse tree. Other encodings include acyclic graphs [9] and tree-adjunct grammars [10].

A point mutation can randomly change the type of the floating-point operation (for example, flipping an add operation to a multiply or an add to a system variable), or randomly change the parameter constant associated with that operation (if it is used). The crossover operation recombines two existing equations to form a new equation. To perform crossover, we select a random location in the genome, and copy all operation and parameter values to the left of this point from the first parent and remaining operations and parameters to the right from the second parent.

Symbolic regression has been applied to explicit equations, dynamical systems [11], and invariant equations [12]. Here, we are applying it to a new type of problem: iterated functions.

3.2 Fitness Function

In order to search for solutions to iterated functions using symbolic regression, we need only to modify the fitness function of the algorithm – the metric for how well an equation explains the iterated function.

For the problem $f(f(x)) = x^2 - 2$, we generated data on the parabola $x^2 - 2$. We sampled 200 points uniformly between $x = -5$ and $x = 5$. Additionally, generate a validation set on a wider range between $x = -10$ and $x = 10$. The fitness during evolution is measured on the smaller range while the fitness on the validation set is used for selecting the best solution and testing for convergence on a general solution.

To measure the fitness of a candidate equation, we evaluate the equation twice for each data point. First evaluating $f(x)$, and then evaluating a second time on this result to calculate $f(f(x))$. We then compare the how close the iterated equation comes to the target iterated function $g(x)$ of the data.

There are many ways to summarize the error over the data set. We used the mean absolute error because it is simple and fast to compute. However, other statistics such as squared error or correlation are likely to also work well.

3.3 Verifying Solutions

For this problem, we are looking for an exact analytical solution. Therefore, we want to verify that the final solution we get is symbolically correct to the iterated map.

The validation set helps us to weed out most overfit solutions but there could still be degenerate solutions that appear numerically correct, but do not analytically derive the target map. Ideally, we would check this in the algorithm itself, perhaps as part of the fitness function. However, we found it was sufficient to perform this verification step at the end of evolution.

Many of the iterated functions require difficult simplifications to reduce down to the exact target – for example, the $f(x) = 2\cos(\sqrt{2}\cos^{-1}(x/2))$ solution described above. We used the Sage Mathematics Software [13] to simplify the iterated solutions.

3.4 Experimental Setup

We used the symbolic regression algorithm [14-16] to search for free-form solutions to the iterated function $f(f(x)) = x^2 - 2$.

We used the deterministic crowding selection method [17], with 1% mutation probability and 75% crossover probability. The encoding is an operation list acyclic graph with a maximum of 32 operations/nodes [9]. Single-point crossover exchanges operations in the operation list at a random split. The operation set contains addition, subtraction, multiply, divide, exponential, logarithm, sine, and cosine operations.

We distributed the symbolic regression evolution over 8 quad core computers (32 total cores) using the island distributed computation method [18, 19]. The island model partitions the

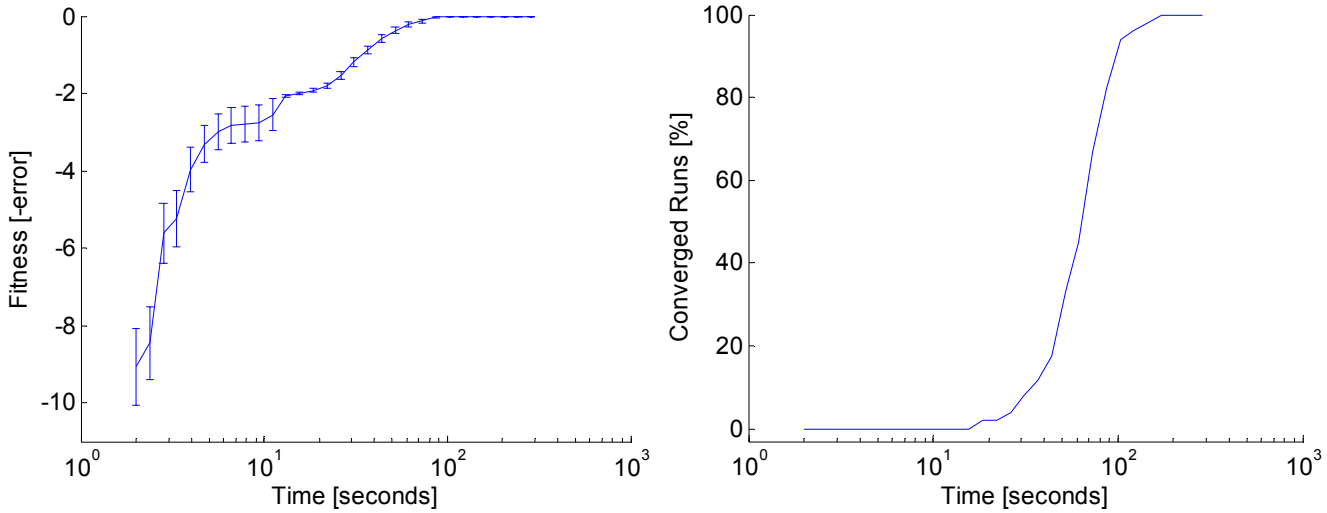

Figure 1. The validation fitness of the most fit iterated equation (left) and the percentage of runs that found an exact solution (right) over the running time of the algorithm. The evolutionary runs converge onto a near perfect fitness solution after approximately two minutes. Results are averaged over 50 independent trials. Error bars show the standard error.

population of solutions into separated smaller populations residing on each computer (or core).

4. RESULTS

In this experiment, we did not provide the inverse cosine operation to the symbolic regression algorithm. Therefore, the algorithm was forced to search for a completely new solution that has not been previously identified, assuming one does exist.

In our first evolutionary run, the algorithm quickly converged onto an exact solution. Figure 1 shows the validation fitness of the highest fit solution over 50 evolutionary runs. The evolution converged onto a perfect fitness solution (epsilon error) within 130 seconds (approximately 2 minutes). This solution is:

Fitness	Solution
-0.620	$\frac{1.037e7x^2 + 2.274e8}{3.709e7x^2}$
-0.460	$\frac{-2.997e3x^4 + 1.553e5x^2 + 1.056e7}{1.245e6x^2 + 2.828e5}$
-0.329	$\frac{2.504e5x^4 + 7.503e6x^2 - 6.404e6}{1.094e6x^2 + 9.509e6}$
-0.129	$\frac{-2.505e8x - 4.664e5}{6.727e10x^3 + 1.252e8x}$
-0.089	$\frac{-1.197e8x - 8.649e3}{8.288e11x^3 + 5.986e7x}$
-0.057	$\frac{-2.045e8x - 2.025e3}{1.033e13x^3 + 1.022e8x}$
-0.035	$\frac{-1.022e4x^2 + 2.045e4x + 0.2025}{5.167e8x^4 - 1.033e9x^3 - 1.022e4x}$
-0.013	$\frac{-9.921e6x - 2}{2.460e13x^3 + 3.968e7x^2 + 4.960e6x}$
-0.009	$\frac{-2.702e7x - 1}{3.652e14x^3 + 8.107e7x^2 + 1.351e7x}$
-0.007	$\frac{-7.207e7x - 1}{2.597e15x^3 + 5.765e8x^2 + 3.603e7x}$
-7.896e-7	$\frac{8.115e13x - 1}{3.293e27x^3 - 4.057e13x}$

Figure 2. A sequence of solutions during an evolutionary run. The most fit solutions at several times are shown in order from top to bottom. Higher fitness (less negative) is better. A general structure is found quickly, and a parameter constant then grows to infinity to converge on an exact solution. Equations have been factored symbolically from their raw encoding.

$$f(x) = \frac{b(b-2ax)}{ax(b-2ax^2)}$$

where $a = 1.16871 \cdot 10^{18}$ and b = 0.683913 are parameter constants.

It is striking that the parameter *a* is so large in this solution. It is very suspicious that an exact solution would really use such a large parameter constant. In fact, our initial thought was that the evolutionary algorithm had found a way to exploit the floating-point round-off in the hardware. Additionally, if we compose this solution with itself, the result is a rather complex rational function – not simplifying neatly to the exact $x^2 - 2$ of the problem. It appeared that the solution was degenerate; perhaps numerically correct but not analytically.

However, we noticed a peculiar trend in the logs of the evolutionary run. Figure 2 shows the sequence of solutions during the run leading up to the final solution. The sequence shows that the algorithm identified the basic structure of this solution early on, and then gradually, solutions evolved to increase the parameter *a* incrementally. We repeated the evolution a number of times; each time getting a similar result.

The parameter *a* appears to stop increasing at approximately 10^{18} due to the floating-point precision of the computer. Increasing it further causes the floating-point calculations performed for this solution to produce *NaN* error codes. Perhaps the equation would become more accurate however with higher precision arithmetic.

Based on this, we suspected that the evolution was attempting to push this parameter constant to positive infinity – much like taking a limit. By taking this limit symbolically, we get the following result:

$$f(x) = \frac{b(b-2ax)}{ax(b-ax^2)},$$

$$f\left(f(x)\right) = \frac{b\left(b-2a\left(f(x)\right)\right)}{a\left(f(x)\right)\left(b-a\left(f(x)\right)^2\right)},$$

$$\lim_{a\to\infty} f\left(f(x)\right) = \lim_{a\to\infty} \frac{b\left(b-2a\left(f(x)\right)\right)}{a\left(f(x)\right)\left(b-a\left(f(x)\right)^2\right)},$$

$$\lim_{a\to\infty} f\left(f(x)\right) = x^2 - 2,$$

Therefore, we verified this is an exact solution – both symbolically and numerically – to the iterated function we want to solve by taking the limit.

Investigating further, we find that the limit of this function is independent of the value of the parameter *b*; it drops out after the second derivative of the numerator and denominator during the limit calculation. However, *b* must be non-zero, otherwise *f(x)* = 0 trivially. Therefore, we refine our solution further by setting *b* = 1:

$$f(x) = \frac{1-2ax}{ax(1-2ax^2)}$$

where the limit of *a* is taken to positive infinity.

This appears to be the first non-trigonometric solution to this problem discovered thus far, and the only other solution yet reported in the literature.

5. CONCLUSIONS

Iterated functions arise in many scientific fields, yet few tools exist to analyze them or find their solutions. We have proposed genetic programming as a method to find free-form solutions to iterated function problems. This approach is applicable to arbitrary problems, and does not require deep mathematical insight into each particular family of iterated functions.

We demonstrated this approach on the notoriously difficult iterated function problem of finding $f(x)$ given $f(f(x)) = x^2 - 2$. Based on the evolved solution for this problem, we were able to identify a novel solution to this problem. The solution composed was verified to be both numerically and symbolically exact.

Our results suggest that genetic programming may be a valuable tool for finding different solutions that do not rely on specific solution forms for arbitrary iterated function problems.

6. ACKNOWLEDGMENTS

Thanks to Brian Josephson (Cambridge University), Richard Palmer (Duke University), and Tom Witten (University of Chicago) for suggesting this problem and helping track its history. This research was supported by the U.S. National Science Foundation Graduate Research Fellowship Program, and U.S. National Science Foundation Grant EFRI 0735953.

7. REFERENCES

[1] R. Brown, "On Solving Nonlinear Functional, Finite Difference, Composition, and Iterated Equations," Fractals, vol. 7, pp. 277-282, 1999.

[2] B. A. Brown, A. R. Brown, and M. F. Shlesinger, "Solutions of Doubly and Higher Order Iterated Equations," Journal of Statistical Physics, vol. 110, pp. 1087-1097, 2003.

[3] "Mathvn journal problems," in Mathvn. vol. 01/2009 mathvn.org, 2009.

[4] B. Andrzej and S. Barbara, "Finding an iterated function systems based representation for complex visual structures using an evolutionary algorithm," MG&V, vol. 16, pp. 171-189, 2007.

[5] A. Bielecki and B. Strug, "An Evolutionary Algorithm for Solving the Inverse Problem for Iterated Function Systems for a Two Dimensional Image," in Computer Recognition Systems, 2005, pp. 347-354.

[6] J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA, USA: MIT Press, 1992.

[7] M. Ben, J. W. Mark, and W. B. Geoffrey, "Using a Tree Structured Genetic Algorithm to Perform Symbolic Regression," in First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA. vol. 414, 1995, pp. 487-492.

[8] D. Edwin and B. P. Jordan, "Multi-Objective Methods for Tree Size Control," in Genetic Programming and Evolvable Machines. vol. 4, 2003, pp. 211-233.

[9] M. Schmidt and H. Lipson, "Comparison of tree and graph encodings as function of problem complexity," in Proceedings of the Genetic and Evolutionary Computation Conference, London, 2007, pp. 1674-1679.

[10] X. H. Nguyen, R. I. McKay, and D. L. Essam, "Solving the Symbolic Regression Problem with Tree-Adjunct Grammar Guided Genetic Programming: The Comparative Results," in The Australian Journal of Intelligent Information Processing Systems. vol. 7, 2001, pp. 114-121.

[11] J. Bongard and H. Lipson, "Automated reverse engineering of nonlinear dynamical systems," Proceedings of the National Academy of Sciences, vol. 104, pp. 9943-9948, 2007.

[12] M. Schmidt and H. Lipson, "Distilling Free-Form Natural Laws from Experimental Data," Science, vol. 324, pp. 81-85, 2009.

[13] W. Stein, "Sage Mathematics Software (Version 3.4)," The Sage Development Team, http://www.sagemath.org/, 2009.

[14] M. D. Schmidt and H. Lipson, "Coevolution of Fitness Maximizers and Fitness Predictors," in Proceedings of the Genetic and Evolutionary Computation Conference, Late Breaking Paper, 2005.

[15] M. D. Schmidt and H. Lipson, "Co-evolving Fitness Predictors for Accelerating and Reducing Evaluations," in Genetic Programming Theory and Practice IV. vol. 5, 2006, pp. 113-130.

[16] M. D. Schmidt and H. Lipson, "Coevolution of Fitness Predictors," IEEE Transactions on Evolutionary Computation, vol. 12, pp. 736-749, Dec 2008.

[17] S. W. Mahfoud, "Niching methods for genetic algorithms," University of Illinois at Urbana-Champaign, 1995.

[18] F. Francisco, S. Giandomenico, T. Marco, and V. Leonardo, "Parallel Genetic Programming," in Parallel Metaheuristics, 2005, pp. 127-153.

[19] G. Christian, P. Marc, and D. Marc, "A Robust Master-Slave Distribution Architecture for Evolutionary Computations," in Genetic and Evolutionary Computation Conference Late Breaking Papers, 2003, pp. 80-87.