RANDOMLY GENERATING WELL-FORMED POSTFIX EXPRESSIONS

applying biological processes to computer programming

Allen Ng
Computer Science Department
University of Wisconsin--Parkside
Kenosha, WI 53141
ng@cs.uwp.edu

ABSTRACT

A model of biological inheritance is applied to develop a random search algorithm which yields an optimal solution. The underlying biology is presented, and application is made to the design of the software. The genetic algorithm's search speed is compared to a purely random search and shown to be much faster at producing well-formed solutions despite having no information on the criteria for correctness.

1.0 INTRODUCTION AND BACKGROUND

Of all systems in existence, none are as optimized and efficient as those found in nature. Indeed, the human genome is so advanced that even with all our efforts we have only been able to observe and record it; no method to exactly duplicate its functionality has been discovered as yet. While human cloning may still be the stuff of science fiction, there are some observations to be made that can be applied to the technology of today. First, a brief primer on the underlying biology.

1.1 Chromosomes

Until recently, the young science of genetics was virtually unheard of outside the domain of a few, highly specialized scientists. Today, some high-school students can tell you that the chromosomes contained in cells are the building blocks of life. Chromosomes are made up primarily of Deoxyribonucleic Acid, or DNA. The nucleic acid portion of DNA comes in the form of nucleotide bases that encode the information for making the proteins necessary for life.

1.2 Mutation and Synapsis

As cells reproduce, they must obviously make more DNA. Our bodies contain trillions of cells and, though at any one time only a portion of them are reproducing, the amount of DNA being replicated at any one time is still enormous. With so much activity going on in a system so complex, mistakes are inevitable. As car salesmen are fond of pointing out (usually while having you apply for extended warranties): if you make 100,000 copies of anything, a lemon is bound to slip through.

DNA is no exception and can be damaged or in other ways altered by many factors most often while replicating. It is important to note that the exact location of these alterations within the chromosome is totally random. As a result, the effects of these alterations are highly unpredictable. Because of redundancies within the DNA sequence, many changes can occur that go unnoticed as they have no net effect. Given that life depends so heavily on the enzymatic action of proteins and that DNA encodes the instructions for producing those proteins, alterations to the DNA sequence that do produce an effect are usually lethal. From time to time however, by chance, an alteration to the DNA sequence, or mutation, can result in a beneficial effect.

These mutations are important as they provide variety within a species. If DNA were always replicated exactly and with no errors and no variation, then every organism within a species would be identical. More important than the damping effect this would have on social gatherings, this would equate to placing all of one's genetic eggs in one environmental basket. Anything in the environment that affects one of them, affects all of them. This is a recipe for extinction.

So important are these sequence mutations, that organisms frequently do it on purpose. Organisms that reproduce sexually produce sex cells, called gametes, during a process called meiosis. The gametes contain the parent's genetic contribution to their offspring. During meiosis, an event called synapsis occurs which partially re-arranges the DNA sequence within the gametes resulting in a wide variety of DNA sequences that could be passed on to offspring.

1.3 Survival of the fittest

It is the DNA that is passed on to offspring that determines the offspring's traits. Though previously stated, it is important enough to reiterate: without mutation or synapsis, an organism's offspring would only contain those traits present in itself. It is these alterations in the DNA sequence that allow for occasional improvements to occur. Be it a beak that is more suited to nut cracking or bacteria that are more resistant to toxins (or medicines, depending on your point of view), DNA mutations that help rather than hinder an organism, help to ensure the survival of the organism so that it may reproduce and pass on the mutation to further generations. This weeding out of weaknesses and preserving of strengths is known as survival of the fittest.

2.0 APPLICATION AND DEVELOPMENT

Our goal here will be to apply the genetic model to a computer program in order to develop an algorithm which yields an optimal solution. For simplicity, we will use randomly generated, arithmetic expressions with the intent of generating well-formed, postfix expressions (if you are unfamiliar with postfix notation, see note at end).

2.1 Solution encoding

First, a method of representing the solution as a linear sequence, or chromosome, must be found. Postfix expressions are linear, the methods of manipulating them are well understood, and most importantly, it is very easy to assign a numerical value to them; namely, by evaluating them. Therefore they are well suited to this task. For our purposes, we will deal with expressions containing the digits 0-9 and the arithmetic operations of addition (+), subtraction (-), multiplication (*), division (/), and taking a modulus (%). In an organism's DNA sequence, there is typically a large percentage of the sequence that does not contain information for protein production. These are referred to as non-coding sequences. Non-coding characters are also allowed in our expressions and, as we have chosen arithmetic expressions to model a chromosome, the letters A-Z will be used as non-coding characters. It should be noted that a digit or operation could conceivably be a non-coding character as well, if it doesn't make sense in the expression. For instance:

+ 2 2 + *

contains two non-coding characters (namely, the first "+" and the "*") neither of which is a letter.

It is important to note that since we are randomly generating the initial expressions, our evaluation of these expressions needs to be very fault tolerant. In other words, the presence of non-coding characters will not deter us from evaluating the expression and assigning it a value. Non-coding characters will simply be ignored.

2.2 Random variation and cross-over

The choice of using arithmetic expressions as our chromosome is of benefit here since making random mutations to the sequence is almost trivial. When the sequence is cloned, there is a chance that a mutation will occur, resulting in a random character (a digit, operation, or letter) being substituted into the sequence. Two expressions are also allowed to "mate" to generate a new expression consisting of randomly selected sub-sequences from each of the parent's sequences. This process is analogous to the synapsis that takes place during meiosis.

2.3 Solution evaluation and fitness testing

Survival of the fittest predicts that those organisms that are the most fit, or have the most advantages, have a higher probability of surviving long enough to reproduce. By reproducing, an organism is able to pass on its traits which, in turn, gives future generations the same advantages and allows the cycle to repeat. However, not all traits are advantageous; only those traits that help an organism to better adapt to its environment represent true advantages.

In order to weed out the weak and preserve the strong, some method is needed to rank our expressions and determine which will pass on their strengths to further generations. An obvious approach is to rank the expressions according to their evaluated value. This does not quite work, however, as a bit of thought reveals that expressions with a large value are not necessarily well-formed (and that is our ultimate goal). We need an environment that does not reward simply ever larger values. To that end, we will define an expression's fitness to be its evaluated value divided by the length of time it took to process the expression.

expression value / time to evaluate

Each chromosome is also examined to determine how well-formed it is. A chromosome's correctness score is calculated by:

# of operands / # of operations+1

A well-formed expression will have 1 more operand than operation giving a correctness score of 1. Each non-coding character encountered during expression processing is deducted from either the number of operands or operations as appropriate.

Our choice of a correctness score also reinforces our choice of a fitness score. While proof of this is outside the scope of this paper, it is intuitively obvious that a well-formed expression should have a higher fitness score than a mal-formed one. This is evidenced by the observation that a non-coding character will add to the expression's processing time while not adding to the expression's value (recall how we defined fitness).

Only the most fit of the candidates in each generation are allowed to reproduce (no, this isn't *1984* or *Handmaid's Tale*...they're only arithmetic sequences). The expressions in the top 25% produce 4 offspring each: 1 from mating with a randomly generated chromosome and 3 from mating with the next fit-ordered chromosome.

It is tempting to want to combine the correctness score into the overall fitness score, but not doing so turns out to be more appropriate. Our goal is to use a model of biological inheritance to guide a random search for an optimal solution. We do not wish to simply compare randomly generated expressions until we match a predefined pattern. By leaving the correctness score out of the fitness calculation, we are assured that this is so.

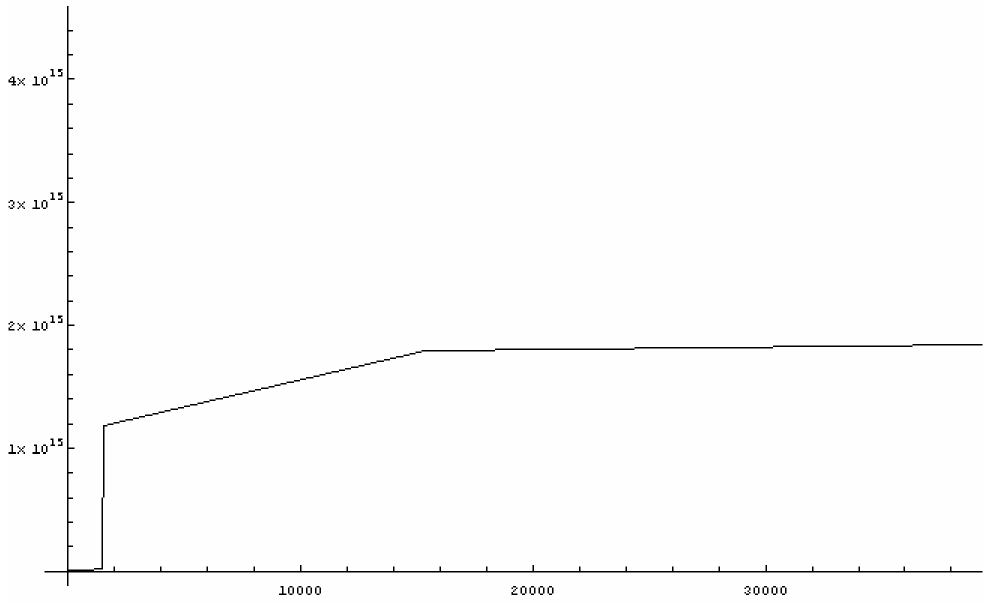

Figure 3.1.1 Fitness scores for successive generations of randomly generated postfix expressions

3.0 ANALYSIS AND SUMMARY

3.1 Results

Using the genetic model as defined above, successive generations of inherited expressions are observed alongside a control group of expressions generated randomly (without inheritance). The figures shown display a plot of value vs. number of generations, with the fitness associated with a particular generation being the highest fitness observed up to that generation.

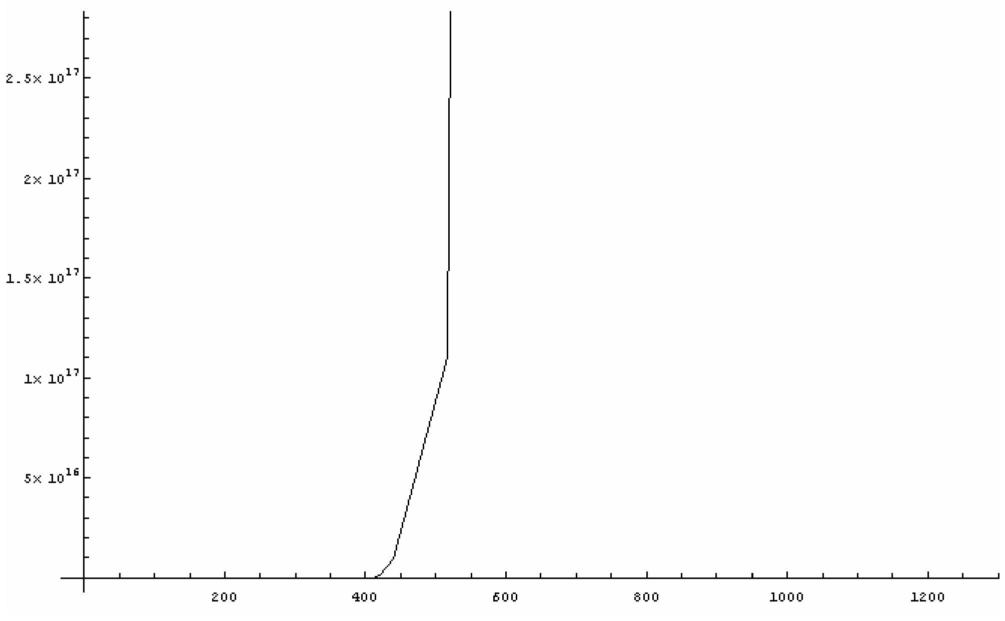

Figure 3.1.2 Fitness scores for successive generations of inherited postfix expressions

3.2 Observations

3.2.1 Fitness scores of inherited expressions compared to random expressions

The most obvious observation is to note that the comparison of fitness scores from randomly generated expressions to those from inherited expressions shows a drastic disparity after fewer than 600 generations. In fact, it takes the randomly generated expressions over 10,000 generations to reach the fitness level of inherited expressions from around 400 generations.

Most importantly however is to note that inherited expressions tend toward being well-formed. Figure 3.1.3 shows that even as the fitness scores of randomly generated expressions increase, their correctness scores do not necessarily follow suit. Figure 3.1.4

shows that inherited expressions do indeed tend towards being well-formed (see section 3.2.3).

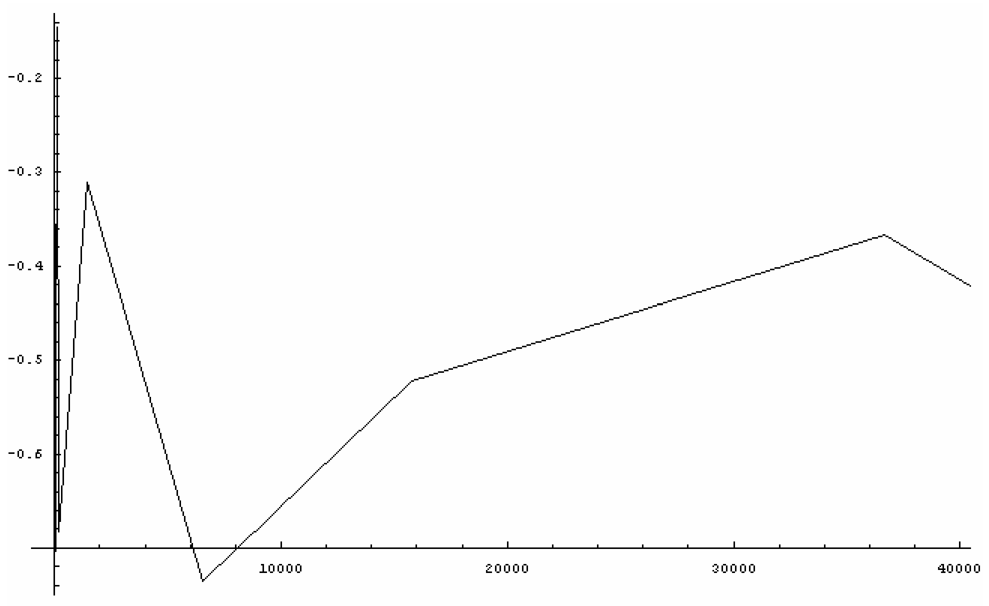

Figure 3.1.3 Correctness scores for successive generations of randomly generated postfix expressions

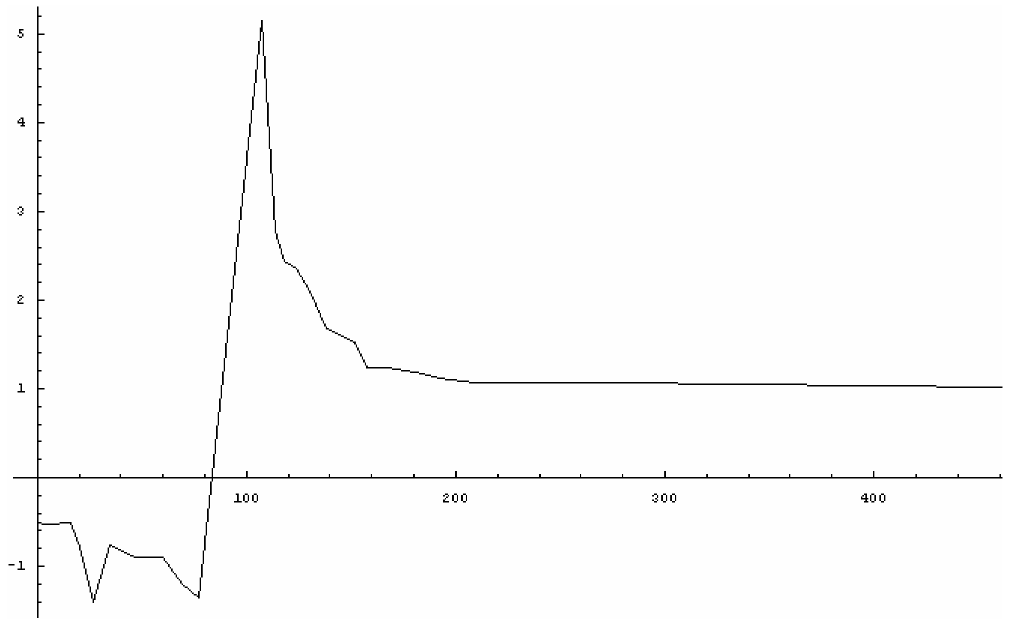

Figure 3.1.4 Correctness scores for successive generations of inherited postfix expressions

3.2.2 Emergence of recurring sub-expression sequences within a population

Examining the actual sequences from successive generations shows recurring sub-sequences being passed on from generation to generation. This appeals to a sense of symmetry with the biological model, but more importantly shows that sub-sequences that improve overall fitness are more likely to get passed on and sub-sequences that detract from fitness are more likely to get passed over.

3.2.3 Slight alterations in configuration lead to drastic variations in patterns

Some of the primary parameters of the program (and their values used for this study) include:

- sequence length (100 characters)
- mutation rate (1%)
- cross-over frequency (40%)

These parameters have shown to be very sensitive in that slight alterations in value can lead to drastic variations in output. This is a hallmark of chaotic systems, a subject worthy of treatment in itself.

One example is mutation rate. It was noted that inherited expressions tended towards being well-formed. This observation requires qualification however, as this property seems to depend on a low, but still greater than 0%, mutation rate. For instance, if the mutation rate is changed to 10% or 0%, the inherited expressions no longer tend toward correctness.

3.3 Summary

Our aim here was to show how the application of a biological model to computer programming can produce a random algorithm that yields an optimal solution (or at least tends to one). This problem was intentionally simple so that focus could be given to the process. The way is now clear to apply the process to more complex situations.

As with all science, this success leads to more questions.

1. What types of problems are suitable for a genetic inheritance approach?
2. What is the relationship between mutation rate and form convergence?
3. Are genetic algorithms more efficient than purely random searches? Than hard-coded techniques? What are the trade-offs?

3.4 Further reading

- Goldberg, David. Genetic Algorithms in Search, Optimization, and Machine Learning. Boston: Addison Wesley, 1989.
- Moriarty, David and Risto Miikkulainen. "Discovering Complex Othello Strategies Through Evolutionary Neural Networks." <http://nn.cs.utexas.edu/downloads/papers/moriarty.discovering.pdf>.

A NOTE ON THE DATA

The reader may have noted a potential loss of precision due to our comparison of floating-point numbers. The software for this research was written in Java. Java's *double* provides around 16 decimal digits of precision. The lengths of the expressions used in this research generate values on the order of 10^{17}. This enables us to resolve values that differ by at least a factor of 10. Observation of the raw data suggests that values in the range used in this research are not greatly affected by this.

Further, the graph in figure 3.1.2 seems to suggest that the inherited expressions improve at an exponential rate. This requires a bit of clarification. The data points included in the graph were the highest fitness scores observed up to that generation. The last point on the graph is merely the highest fitness score observed from all generations.

POSTFIX NOTATION

The following arithmetic expression should be familiar to most:

5 – 2

This expression uses what is called *infix* notation, where the operation is *in*-between the operands. *Postfix* notation is nothing more than swapping the position of the operator so that it appears *post*-expression like so:

5 2 –

The advantage of using postfix rather than infix on a computer is that postfix needs no parenthesis. As an example, take:

5 – 2 * 4

Depending on which operation you wanted to do first, parenthesis might be required as in:

(5 – 2) * 4

With postfix, no such ambiguity exists. If the subtraction is to be performed first, we have:

5 2 – 4 *

and if the multiplication is to be performed first, we likewise have:

5 2 4 * -

Just like infix, postfix can become malformed. For example:

2 + 3 4

contains too many operands and so is not well-formed infix. Likewise:

+ 2 2 + *

is not well-formed postfix. In general, for a postfix expression to be considered well-formed, it must contain 1 more operand than operation and there must be two operands prior to any operation.