AI and the Primordial Soup

Walter B. Gress V

Drexel University

Friday, June 7 2013

This article discusses a random algorithm to create emergent behavior by cross-infecting virus programs, mutating them with a genetic type algorithm, in hopes to generate over time a more complex program.

# introduction

Often times in Computer Science and Mathematics we study deeply the world around us to extract behaviors, models, and algorithms. We have been extremely successful in those practices. In some practices, we may fail miserably, but more often that naught, we are able to glean algorithms or models from these studies. Studying simple things in nature such as birds, termites, ants, bees, wasps, genetics, the human brain, cellular automata, artificial immune systems, plant structures, and even disease.

So, we have multiple theories that are somewhat related to each other. We know from studies of evolution and what we think that the beginning of life on this planet is that Earth started out with enough water for pools of water to exist. These pools of water had various inorganic components which eventually began to combine in different ways and eventually resulting in RNA and then it seems DNA.

One of the issues is which came first, virus or bacteria? Well viruses cannot reproduce on their own, meaning, there had to be bacteria around during or before viruses came into being. There is a problem here. Either a) Viruses mutated to the point where they “fell apart” from being bacteria, b) Both bacteria and viruses evolved simultaneously, or c) Viruses evolved from Bacteria – which is shown not to be true since viruses cannot reproduce. Clearly from an evolutionary perspective something doesn’t’ seem right here. Life evolves over time, improving in most cases, with mutations driving the progress over time. Sometimes the mutations are beneficial, those with those mutations have an advantage have “evolved.” Those who are weaker or are the same tend to die off.

Complexity wise and on average, bacteria are far more complex than a virus. A large number of virii are single stranded, RNA (no DNA, no double helix). I can’t help but contradict the idea that viruses didn’t exist before bacteria just because they might not have any food. I feel this is a fair position to take.

# APPLICATIONS TO COMPUTER SCEINCE

Now about Computer Science. We have established that a small number of random sets of chemicals, gradually pieced themselves together in a way that would be functional and optimal to bring forth the components of life, which combined in a manner to develop the first cell and so on.

I posit that once again Computer Science can benefit from nature. I’d like to ask what is the smallest operating program on a computer. And how can that small program mimic the smallest organic life? How can that small program connect to other small programs or change them; how can we force evolution on a series of programs: A series of very small programs. Let’s call those programs viruses.

I say we develop small, mission-oriented, viruses to infect each other with various keywords. For every iteration, a parser runs on the virii to see if any of them have randomly mutated so that they will parse. That is, it checks to see if one program has changed one program to a new program.

# CONDITIONS

Let the following conditions holds:

* Let there be a complexity factor c.
* Let n dictate the number of lines of code
* Let the parsing score equal p.
* Let f, the failure score be a statistic on how often an infected program fails
* Let t, the success score on how often an infected program works properly.
* Let k be the keyword substitution operator. K exchanges one keyword for another.
* Let q be the number of keyword infections this round.

Then I conjecture the following relationships:

* In order for the program to **execute**, f < t.
* In order for the program to **execute**, p >= f
* In order for the program to **evolve,** n0 > n1 where n0 is the number of lines prior to infection and n1 is the number of lines post infection.
* In order for the program to **evolve,** c0 > c1 where c0 is the complexity prior to infection and c1 is the complexity post infection
* In order for the program to **evolve**, q > 1, k(word) = k’(word).

# PROOF

Given multiple mutations, a set of viruses mutating each other will evolve.

Base:

Given the above relationships are true and we have two viruses a and b, let k(a,b) = k’(a,b), where a is an element of the first virus and b is an element of the second virus, function k will swap a and b and mutation will occur, specifically, mutation.

Induction:

Let there be k viruses, each with nk amounts of program code. Consider one time slice. Let there be a0, a1, a2, …, ak number of code segments to mutate. The viruses pair off and k(ai, aj) = k’(ai, aj) for all i,j. If f<t and p>0 the mutation

# ALGORITHM (for two)

1. Read in program code for virus a and virus b.
2. Pull a keyword from a keyword database.
3. Randomly swap keywords in virus a with virus b code.
4. Introduce a new keyword and swap with virus a and virus b.
5. Parse the code for virus a and virus b to make sure it is valid.
6. Repeat.

# CONCLUSION

We conjecture through a simple replacement and validation process, we can take multiple program codes and over an exceptionally large number of iterations, we can see more and more random and complex programs generated.