

Luis Rocha's Agent-Based Model of Genotype Editing

Joe Blaylock

January 10, 2008

1 Introduction

The simple genetic algorithms of a few decades ago have given rise to an entire suite of research tools and algorithms which comprise the new field of Artificial Life (Alife). In the intervening years, Alife has grown and blossomed, producing valuable insights for researchers in cognitive science and artificial intelligence. However, Luis Rocha and others contend that it is still too married to the biological paradigms of yesteryear. They say that greater insights into biological processes are available if researchers adopt more modern ideas about genomes and phenotype expression.

While these new, biologically plausible algorithms offer much faster and

more robust convergence to solution in “evolutionary time¹”, it remains to be seen whether they offer an advantage over traditional genetic algorithms in terms of CPU time. Nevertheless, they are immensely exciting. They can help guide our intuition about biomimetic computational systems in general, but also, by modeling current molecular biological theories, they help us to understand life itself.

2 Genetic Algorithms

2.1 Theoretical Assumptions

The biological model of early Alife is one which holds that DNA is for making proteins. Any DNA which doesn't make proteins, is 'junk'. The marvel of multicellular life is that so much of our DNA can be 'junk', and yet we still seem to function alright. The various RNAs, transcribed out of DNA by RNA polymerases, function primarily to copy the DNA strand (including the portions carrying the RNA) during cellular reproduction, and to help transcribe the DNA into protein sequences within the cell.

¹In which we count generations, rather than computer clock cycles.

In this way of looking at biology, the relatively limited role of RNAs has several practical ramifications. One is that, because they act merely as interpreters of the protein-production information contained in DNA, they're boring. If a researcher or modeler wants to understand the complexity of instantiated organisms (or phenotypes, as they're called), they can largely ignore RNA and the entire ontogenetic² process. Understanding the mappings between DNA base pairs and the proteins that they produce can tell you everything you need to know about an organism. Of course, understanding these mappings turns out to be quite difficult in practice, but we'll come to that in Section 3.1.

Another important point about the primacy of protein-coding as the proper role of DNA is that it makes biological complexity mysterious. Organisms with far less biological complexity than human beings, such as nematode worms, devote far more of their DNA to protein coding. Even with a much shorter genome, such a high proportion is allocated for protein coding, that they actually end up having more protein-coding genes than we do. If protein coding is the primary function of DNA, and if RNA is just a dumb interpreter

²That is, the process of instantiating a genotype — a collection of description of proteins — into a phenotype — an organism in the world.

of that DNA into proteins, then there must be something very special about the proteins coded in our DNA, that we are able to live with so few of them. That, or else DNA can compress information magically well, so that all of our biological complexity is summed up in the very small part of our DNA that's not 'junk'. Of course, we now know that this is silly, for reasons that will be outlined in Section 3.1.

2.2 How They Work

The extrapolation from this point of view on genomes to traditional genetic algorithms should be fairly straightforward. Despite the lofty goals Rocha attributes to Langton[2], early genetic algorithms work was happy to achieve limited success as an optimization or search strategy³, and so took a least-effort approach to implementing the theoretical framework we've just discussed. Problems consist of finding some sequence of bits which suitably solves an equation. The mapping of environmental parameters into parts of the bit string, when necessary, and of mapping the bit string into a solution, is left to the human researcher. To find the bit string, a genetic search does

³One of the canonical genetic algorithms texts of the early days of the field is David Goldberg's *Genetic Algorithms in Search, Optimization, & Machine Learning*. The title says it all.

a random walk over the parameter space defined by the string.

At each generation⁴, the best solutions are chosen to reproduce. In some implementation, pairs of the best solutions are “mated” with crossover (that is, substrings are transposed between selectees). Other times, the best solutions are simply copied into the next generation, in numbers proportional to their closeness to a solution. In any case, occasionally mutations are introduced, which are implemented as stochastic bit-flipping in the string, usually with a very low probability.

Traditional genetic algorithms tend to work well in situations in which the optimization surface is relatively smooth. Because the fitness-proportionate selection tends to implement a kind of hill-climbing, it is easy for them to get ‘caught’ in local maxima and not improve further. Crossing regions of lower fitness to reach the further shores of higher-fitness values generally requires high mutation rates or other artificial constraints that prevent having too many of the same genotype. On the other hand, because they don’t simulate the development of instantiated organisms from their genetic code, they can usually be implemented in a relatively computationally inexpensive way.

⁴Defined as one iteration of the production and evaluation of some fixed-size collection of strings

3 Agent-Based Models of Genotype Editing (ABMGEs)

3.1 Theoretical Underpinnings

As discussed in Section 2.1, the role of the non-protein-coding RNA portions of DNA strands (ncRNA) was overlooked for some time. Eventually, however, studies such as Taft et al[4] showed that there was a significant link between the amount of ncRNA (previously referred to as ‘junk’) in a genome, and the biological complexity of the phenotype. The explanation put forward, and later confirmed by experiment, was that the ncRNA was actually acting in a non-transcriptional way on the DNA strand — it was doing more than merely interpreting. If RNA could modify the products of the interpretation of a DNA strand, then a given sequence of protein-coding instructions would be capable of performing a much wider variety of actions, depending on what ncRNAs worked on it.

This meant that we suddenly had a theoretical way to understand the complexity of instantiated organisms in terms of context-dependent modification of the instructions from their DNA, during organismal development.

This meant that organisms could not only be more complicated than we would expect from the number of proteins directly coded for in their DNA — because now any given coding could actually produce many different proteins — but it gave a mechanism whereby the story of an organism’s development could effect its gene expression. Complex organisms could be more flexible with this new way to adapt to environmental factors.

For a metaphor, recall that in Section 2.1, we referred to RNAs as interpreters of the instructions in the DNA. This new view of ncRNAs, then, upgrades the interpreter to have features of reflectivity⁵. DNA went from being essentially a dead data structure, to being a powerful piece of living code, in a richer language than we had ever expected.

3.2 The Model

Rocha’s ABMGE seeks to model this new understanding of ncRNA in a simple software system. By doing so, he can help to educate our intuition about the nature of genome editing, its purpose and utility. Further, he can explore new approaches to evolutionary search and optimization, and

⁵Reflective languages allow programs to be modified by themselves at run-time. Cf. Wikipedia:Reflection[7].

evaluate their effectiveness objectively.

In caricature, Rocha’s model resembles the sketch of the traditional genetic algorithm given in Section 2.2. The main difference comes at two points in the cycle: Rocha introduces a phenotype instantiation step just before genotype evaluation, and he adds an extra bit of crossover just after the selection phase. He divides the genome into two parts, the “Codotype” and the “Editype”. When it is time to instantiate the genotype to be evaluated, the editype is *applied* to the codotype (in a manner to be described below) to generate a particular instance of the genotype (a phenotype). Then, when a particular genotype is selected for reproduction, it undergoes crossover multiple times: once for the codotype, and once for each editor encoded into the editype. In this way, the editors evolve along with the material they edit.

But what are these editors? An editor is a tuple consisting of a string, a function, and a concentration parameter. When the editype (a collection of editors) is applied to the codotype, each editor is applied in turn. When an editor is applied to a codotype, then the codotype is searched for the editor’s string. If a match is found, then the editor’s function is applied stochastically, with probability equal to the concentration parameter.

This is the ABMGE Algorithm⁶:

1. Randomly generate an initial population of l agents, each consisting of a codotype (a n -bit string) and an editype (a collection of editors having a string, a function, and a concentration).
2. Edit each agent's codotype S : apply each editor with probability equal to its concentration; if the editor's string matches a substring of S , edit s with the function of the editor.
3. Evaluate the fitness of the edited genotype for each agent.
4. Repeat until l offspring have been created.
 - a. select a pair of parent agents for mating;
 - b. apply codotype variation operators (mutation and crossover);
 - c. apply editype variation operators (mutation and crossover).
5. Replace the current population with the new one and go to step 2.

Because the application of editors to genotypes is stochastic, even multiple instances of the same genotype in the population will tend to produce unique phenotypes. As we will see in Section 3.3, this yields a robust search across a large portion of the parameter space. In any random population, the odds that any single individual will be a good fit to the problem are very small. But the simultaneous evolution of editors for the genotype information allows a large section of the solution space to be searched at once, yielding both robust results and rapid⁷ convergence of the population.

⁶As described in Rocha & Kaur[3]

⁷Rapid in terms of the number of generations from the beginning of the simulation until maximum-fitness-reached. We make no claims about the computational cost of each generation or the consequent wall-clock time to convergence.

3.3 Performance

Rocha et al tested the performance of his ABMGE in a wide variety of circumstances⁸ including static environments (Small Royal Road, De Jong Function, Optimal Control, Modified Schaffer's Function, and Schwefel's Function) and ones which changed over time (Oscillatory Small Royal Road, Dynamic Schaffer Function, and Dynamic Schwefel's Function). In every circumstance tested, the ABMGEs (either with or without editype crossover) converged to their best performance faster than the reference genetic algorithm, and also came close to optimal solutions far more often.

In the static environments, the ABMGE with editype crossover only faced real competition on the Optimal Control problem and Schwefel's function. In both of these cases, the differences between the performance of ABMGE with editype crossover and ABMGE without editype crossover were statistically insignificant. The Small Royal Road test offers a good intuition as to why this should be: while the GA was evolving its entire genotype towards optimality (a string of all 1's), the ABMGE solutions only had two chances: they could evolve their codotype towards all 's, and also evolve their editype towards

⁸For the gory details, see Huang, Kaur, Maguitman and Rocha[1].

massive 1 insertion. Either — or both — would make the solution optimal.

In the dynamic environments, the ABMGE class of solutions still dominated the genetic algorithm, but the two different ABMGEs (with and without editype crossover) were in much closer competition. In problems where a memory of previous solutions might be a hindrance – such as the Oscillatory Royal Road, the ABMGE without crossover maintained a more diverse gene pool, and clearly outperformed both the crossover version and the genetic algorithm. Both of these had much more serious population crashes when the environment went through its extreme shift. On more subtle dynamic problems, where the changes between extrema come with more warning, the crossover version of the ABMGE comes back into the race, and is statistically indistinguishable from the non-crossover version. Both clearly do much better than the reference genetic algorithm.

We thus find that crossover versus non-crossover of editypes is a tradeoff between exploitation of solutions found versus exploration for new solutions. By employing editors for the codotype, ABMGEs produce a lot of bad solutions; much of the population dies in each generation. This means that solutions closer to optimality can be found faster, however, because more of

the search space is explored. Each generation is a wide exploration around a centroid established by the high water mark of the previous generation. In this way, the ABMGE enables a rapid (in generational time) zeroing-in on good solutions.

4 Future Work

From the algorithm outlined in Section 3.2 we can see that the crossover and mutation of editypes effects the matching strings of editors and their concentrations, not the editing functions themselves. The work so far has been sufficient to show that mutation and crossover of these parameters can enable an agent with a fixed suite of editor functions to take great advantage of them, but what about evolving the editor functions themselves? Multiple times in Huang, Kaur, Maguitman, and Rocha[1], allusion is made to further work in just this area. How much more powerful might these models become if the transformations themselves, rather than merely the circumstances under which they're performed, become evolvable?

References

- [1] Huang, C., Kaur, J., Maguitman, A. and Rocha, L. (2007). Agent-based model of genotype editing. *Evolutionary Computation*, 15(3);In Press-.
- [2] Rocha, L.M., “Reality is Stranger than Fiction: What can Artificial Life do about Advances in Biology?”. Invited presentaton for the “Biocomplexity” discussion section at the *9th European Conference on Artificial Life*, September 12, 2007 in Lisbon, Portugal.
- [3] Rocha, L.M. and Kaur, J., “Genotype Editing and the Evolution of Regulation and Memory”. *Proceedings of the 9th European Conference on Artificial Life*. LNAI, Springer, **4648**: 63-73.
- [4] Taft, R.J., Pheasant M., and Mattick, J.S., “The relationship between non-protein-coding DNA and eukaryotic complexity”. *Bioessays*. 29(3):288-99.
- [5] Ploidy. (2007, December 8). In Wikipedia, The Free Encyclopedia. Retrieved 18:35, December 10, 2007, from <http://en.wikipedia.org/w/index.php?title=Ploidy&oldid=176650902>
- [6] Polyploidy. (2007, December 6). In Wikipedia, The Free Encyclopedia. Retrieved 18:35, December 10, 2007, from <http://en.wikipedia.org/w/index.php?title=Polyploidy&oldid=176225058>
- [7] Reflection (computer science). (2007, December 6). In Wikipedia, The Free Encyclopedia. Retrieved 18:33, December 10, 2007, from http://en.wikipedia.org/w/index.php?title=Reflection_%28computer_science%29&oldid=176225058
- [8] RNA. (2007, December 9). In Wikipedia, The Free Encyclopedia. Retrieved 18:30, December 10, 2007, from <http://en.wikipedia.org/w/index.php?title=RNA&oldid=176877732>