On some emergent properties of variable size evolutionary algorithms

Wolfgang Banzhaf[1] Frank D. Francone[2] and Peter Nordin[13]

[1] LS11, Dept. of CS, University of Dortmund, 44221 Dortmund, Germany
[2] RML Inc., 360 Grand Ave., Oakland, CA, 94610, USA
[3] DaCapo AB, Kronhuesgatan 9, 41105 Goeteborg, Sweden

Abstract. We argue that variable-length evolutionary algorithms behave qualitatively different from their fixed-length counterparts. We point out some of these differences derived from emergent properties of the algorithms.

Evolutionary algorithms with variable-size genomes have emergent properties that may not be obvious until observed in action. As such they share many properties with DNA and RNA in nature. In particular, it has turned out over recent years, that variable-length evolutionary algorithms behave qualitatively different from their fixed-length counterparts which are mainly studied in the EA community.

Differences between fixed and variable length genomes are

1. the ability of variable-size genomes to choose their own representation for the target problem,
2. the emergence of neutral and effective code, the former of which ultimately produced in large amounts in EAs with variable size genomes
3. the different ability of both types of genomes to stabilize a population against the destructive effects of operator activities.

We note in passing that variable size EAs also have a maximum length for genomes. The reason is that variable-length genomes are mapped into a finite storage space. One must, therefore, carefully discern possible boundary effects when length of genomes approaches the maximum from real variable size effects.

1 Choosing a representation

The ability to evolve a representation for a problem depends on the ability of a learning algorithm to modify its own structure that codes for a solution. Typical fixed length EAs have little capacity for evolution of their genotype (their structure) because one has determined the length and meaning of each element in advance. By evolving structure, a variable length genotype may be able to learn not only the parameters of a solution, but also how many parameters there should be, what they would mean and how they needed to interrelate. This variability introduces many degrees of freedom into the evolutionary search that

are missing in fixed length structures. Two examples of variable length EAs are messy GAs [2] and Genetic Programming (GP) [3]. Here we shall have a closer look at the latter.

As a simple exemplification, consider a GP-system which has to breed algorithms with the following arithmetic functions: *Plus, Minus, Times, Divide*. The system might change this representation by ignoring any of these functions, thereby reducing the function set. So if solutions which use the *Divide* operator were, in general, producing worse results than others, we could expect the system to reduce and, eventually, to eliminate the *Divide* operator from the population.

This is but one example showing the ability of variable-size genomes to adapt their representation to the problem they are supposed to solve. But while variable length solutions have potential advantages to artificial evolution, they also appear to be the cause of what may be troubling emergent properties called, variously, *neutral code*, *non-coding segments*, *introns* or *bloat*.

2 Neutral and effective parts of the genome

In GP, an interesting observation made by Peter Angeline [1] was that many of the evolved solutions in [3] contained code segments that, when removed, did not alter the result produced by the solution. He also made the connection between such code and biological introns. Subsequent research has revealed that 'introns' are a persistent and problematic part of the GP process [4, 6]. The evidence is strong that, as in biology, evolution selects for the existence of introns [5]. (For a recent review of biological introns, see [7].) But how and why?

Two features may be used to define what we shall call introns here:

- An intron is a segment of the genotype that emerges from the process of the evolution of variable length structures; and
- An intron does not affect the survivability of the individual directly.

Because introns have no effect on the fitness of the variable size individual, we would not expect to see strong selection pressure to create this genomic structure. After all, it does not affect the fitness of an individual. So why do introns emerge?

In summary, the answer is that, while introns do not affect the fitness of the individual directly, they may affect the likelihood that the individual's descendents will survive. In other words, the *effective* fitness of an individual is a function not only of how fit the individual is *now* but also of how fit the individual's offspring are likely to be *in the future*. By this view, the ability of an individual to have high fitness children (given the existing genetic operators) is as important to the continued propagation of its genes through the population as is its ability to be selected for crossover or mutation in the first place. It does no good to have high fitness and to be selected for crossover or mutation if the children thereby produced are very low in fitness. Thus, we would expect individuals to compete with each other to be able to have high fitness children.

3 Effective fitness

This, naturally, leads to the question whether it would make sense to modify fitness with a term which reflects the influence that an individual exerts onto the fitness of its offspring. Indeed, we believe so. Because crossover is by far the most prominent operator used in GP, let us, for the sake of the argument, concentrate on crossover here. The argument can be easily extended to other search operators.

The probability that crossover in an 'effective' part of a genome will lead to a worse fitness for the individual is called probability of destructive crossover, p^d. Recall now that one could discern absolute length[4] and effective length of programs in GP (and of genomes in other variable length EAs in general). Let L_j^e be the effective length of genome j, and L_j^a its absolute length. Let us further assume only crossover (probability p_c) and reproduction in a generational EA.

The probability that a program j will be destroyed by crossover is

$$p_c p^d \frac{L_j^e}{L_j^a}, \tag{1}$$

i.e. the probability of the application of the operator times the probability of its action being destructive. This probability has a profound effect on the proportion of individuals with this genome [4]. We can use the crossover related term as a direct subtraction term from regular fitness in an expression for reproduction through selection. In other words, reproduction by selection *and* crossover acts as reproduction by selection *only*, if the fitness is adjusted by the term mentioned above. We call this "effective fitness" f_j^e.

The effective fitness of a parent individual, therefore, reads

$$f_j^e = f_j[1 - p_c \cdot \frac{L_j^e}{L_j^a} \cdot p^d] \tag{2}$$

and measures how many children of that parent are likely to be chosen for reproduction in the next generation. A parent can increase its effective fitness by either of two strategies:

1. by lowering its effective genome length (in GP, that is, having its functional code become more parsimonious); or
2. by increasing its absolute genome length

In other words, the difference between effective fitness and actual fitness measures the extent to which the destructive effect of genetic operators is warping the real fitness function away from the fitness function originally designed.

The same argument can be made for other types of operators working on genomes, e.g. for mutation. The main effect of operators can be subsumed in their tendency to *change* genomes to the good or to the bad. The main goal

[4] We use length and size synonymously. In tree-based GP the linear notion of length can be substituted by another complexity measure, e.g. number of nodes.

of genomes, on the other hand, can be subsumed in their tendency to try to proliferate or to *stabilize themselves*, whether their fitness is high or low. It is in the balance of these two processes that evolutionary algorithms work.

References

1. Peter John Angeline. Genetic programming and emergent intelligence. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 4. MIT Press, 1994.
2. D.E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation analysis and first results. *Complex Systems*, 3:493 – 530, 1989.
3. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT press, Cambridge, MA, 1992.
4. Peter Nordin and Wolfgang Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, San Francisco, CA., USA, July 1995. Morgan Kaufmann.
5. Peter Nordin, Frank Francone, and Wolfgang Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 6, pages 111–134. MIT Press, Cambridge, MA, USA, 1996.
6. Terence Soule, James A. Foster, and John Dickinson. Code growth in genetic programming. In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 215–223, Stanford University, CA, USA, 28–31July 1996. MIT Press.
7. A.S. Wu and R.K. Lindsay. A survey of intron research in genetics. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature IV. Proc. of the Int. Conf. on Evolutionary Computation*, volume 1141 of LNCS, pages 101 – 110, Berlin, New York, 1996. Springer.

This article was processed using the LaTeX macro package with LLNCS style