Introduction to the Special Issue: Variable-Length Representation and Noncoding Segments for Evolutionary Algorithms

Annie S. Wu
Naval Research Laboratory
Code 5514
Washington, DC 20375
aswu@aic.nrl.navy.mil

Wolfgang Banzhaf
Department of Computer Science, LS11
University of Dortmund
D-44221 Dortmund, Germany
banzhaf@cs.uni-dortmund.de

The huge variation in the number and types of organisms that have evolved in nature is paralleled by an equally impressive diversity in the underlying genetic representation. Variation can be found, not only in the size of genomes, but also in the organization of information and the structure of the genomes. The result of this freedom of expression is that natural evolution is virtually unlimited in the types and complexity of the organisms that can be evolved. An entire new field has emerged in computer science to investigate the evolution of complexity and how it may be applied as problem-solving tools (Fogel, Owens, & Walsh, 1966; Holland, 1975; Koza, 1992; Rechenberg, 1973). Evolutionary computation systems traditionally employ a much more rigid representation of information than natural systems. Though these simpler, fixed representations may be easier to analyze, they also limit the scope and complexity of what may be evolved. We believe that it is time to seriously investigate the impact of more complex, variable representations on artificial evolutionary systems. The goal of this special issue is to examine the issues involved in extending the representational complexity of evolutionary algorithms (EA) and how this affects an EA's ability to evolve problem solutions. In particular, we focus on the ramifications of introducing variable-length representations and noncoding regions to EAs.

A simple but significant difference between natural and computational evolutionary systems is the prevalence of variable-size representations in natural systems. In nature, genome sizes range from as little as 10^5 base pairs (bp) for algae to as much as 10^{11} bp for some plants and amphibians (Lewin, 1994). Even within phyla, the range of genome sizes varies from twofold in birds and mammals to more than tenfold for plants and amphibians. Variable-size representations lead naturally to the dynamic organization of information (genes) and to the appearance of noncoding or unexpressed segments in the genome. Noncoding DNA or DNA that does not directly contribute to the production of proteins is thought to make up upwards of 90% of some genomes (Bell, 1988; Nei, 1987). Thus, nature is able to vary both the amount and ratio of coding and noncoding regions in a genome, which, in turn, is thought to influence the shuffling or recombination of the coding regions (Gilbert, 1987). A brief review of the genetic process and a discussion of the different types of noncoding DNA is presented by Wu and Lindsay (1996b).

Extending such flexibility to EAs would greatly increase the complexity of the problem solutions that may be generated by these algorithms and would make these algorithms much more challenging to understand. EAs that use variable-size genomes are expected to have more expressive power to solve problems in which the structure and size of a satisfactory solution is unknown in advance. Noncoding segments are thought to provide protection from the disruptive effects of genetic operators, to encourage the recombination of coding regions, as well as to provide natural backups for coding regions. A better understanding of the origin, effects, and implications of variable-length representations and noncoding segments on EAs would allow researchers to use these structures more effectively in practical applications.

EAs cover a number of different types of algorithms, including evolutionary programming, evolutionary strategies, genetic algorithms (GA), and genetic programming (GP). Of these, GP has historically had the most variable and flexible representation. GP started with the evolution of variable-sized LISP S-expressions but has grown to include linear and graph representations of programs that vary in both size and content. Interestingly, with no bound on size, GP tends to evolve programs that are much larger than necessary, containing large sections of code that are never used (Blickle & Thiele, 1994; Koza, 1992). Though many early efforts focused on methods for "editing out" these noncoding regions, or "bloat," later studies indicate that the size of evolved programs may affect their fitness or utility (Langdon & Poli, 1997; Nordin & Banzhaf, 1995; Rosca, 1996; Soule, Foster, & Dickinson, 1996).

Soule and Foster (this issue) examine in detail the relationship between code growth and parsimony pressure (fitness penalty based on program size). They find that GP is able to balance the demands of size versus performance if parsimony pressure is not overapplied. Theoretical analyses indicate that the effects of parsimony pressure on a population can be predicted from the size and fitness of the programs in the population, and suggest that it may be useful to vary parsimony pressure based on variations in program size and performance.

Initially thought to be useless junk, noncoding regions and their potential contributions to the evolutionary process have gained increasing interest (see Chapter 7 of Banzhaf, Nordin, Francone, & Keller (1998)). Haynes (this issue) investigates the use of noncoding regions as backup or scratch space in a GP system. He first confirms that the complete removal of noncoding regions is detrimental to the GP search process, then shows that encouraging the GP to save duplicate copies of coding regions in noncoding regions can significantly improve the performance of the system. Haynes further investigates the exchange and preservation of coding regions among the individuals of a GP population.

Nordin, Francone, & Banzhaf (1996) introduced explicitly defined introns (EDI), a type of noncoding segment, into a linear machine code GP and examined their effects on crossover. Empirical studies found that growth of noncoding regions appeared to be linked to a decline in destructive crossovers. Their overall results indicated that, for a linearly encoded GP system, the explicit addition of noncoding segments can improve the fitness and generalization of evolved programs and can reduce the amount of time required to evolve suitable programs. Smith and Harries (this issue) extend this work on EDIs to tree-based GP representations. This article focuses on the causes of code growth and the contributions of various types of introns. Smith and Harries find that noncoding regions generally have negative effects on the performance of tree-based GP.

Even though variable-length genomes and noncoding regions have been an integral part of GP, they are just starting to emerge in other EA systems that have traditionally used more rigid problem representations. In particular, there has been quite a bit of work on the effects of noncoding regions on a GA. Levenick (1991) was the first to introduce noncoding segments

to a GA. He found that the inclusion of noncoding regions can improve GA performance. Similar studies on a different set of test functions yielded mixed results. Forrest and Mitchell (1992) found no significant improvement in GA performance with the addition of noncoding segments. Wu and Lindsay (1995) found moderate improvement in the stability of a GA when noncoding segments are added.

A significant difference between the noncoding material in the above GA studies and that found in GP is that the noncoding regions in the GA studies were manually inserted rather than naturally evolved. The ability to dynamically adapt the amount and locations of noncoding and coding regions greatly increases the expressive power of a GA. Wu and Lindsay (1996a) experimented with "tagged" coding regions and found that performance improved significantly if the locations of coding and noncoding regions were allowed to evolve dynamically. Mayer (this issue) extends this work with a detailed analysis of both fixed and variable noncoding segments in a GA. Variable noncoding segments were achieved by demarcating coding regions with promoter and terminator sequences or tags. Experimental results indicate that noncoding segments can significantly improve GA performance on select types of problems. Mayer further investigates alternative crossover operators that promote the distribution of building blocks or coding regions among GA individuals.

Few GA systems have used variable-length individuals (Goldberg, Korb, & Deb, 1989; Harvey, 1992). This simple modification to a GA introduces a host of new issues that need to be addressed. Burke et al. (this issue) describe a GA that has been extended to work with variable-length genomes. This work focuses on investigating the factors that affect the evolved size and fitness of individuals and on understanding the use of noncoding regions in a GA. Empirical studies produced clear evidence of regions switching between coding and noncoding status, adding support to the hypothesis that noncoding regions may serve as backups for coding regions by storing extra copies of useful information.

There has been more than enough work on tweaking parameters and problem representations to make an EA to solve a particular problem well. We believe that it is time to start focusing more on understanding the dynamics of these algorithms rather than simply getting them to work on specific problems of local interest. Problem representation is an important component and contributor to EA behavior. The articles in this issue investigate alternative representations and explore the implications of giving some of the decision making about representation to the algorithm itself, i.e., allowing EA representations to evolve and change dynamically. We hope that the ideas presented here will be useful to those who are trying to decide how to set up problem representations for their problems, as well as encourage continued explorations in this area.

We would like to thank the authors who submitted their work and the reviewers who took the time to read and comment on the submitted papers for helping us to put together a very interesting special issue.

Acknowledgments

Annie S. Wu was supported by the National Research Council and the Naval Research Laboratory. Wolfgang Banzhaf gratefully acknowledges support from the Deutsche Forschungsgemeinschaft under grants Ba 1042/5-1 and Ba 1042/5-2.

References

Banzhaf, W., Nordin, P., Francone, F., & Keller, R. (1998). *Genetic programming—An introduction*. San Francisco, CA: Morgan Kaufmann.

Bell, G. I. (1988). The human genome: An introduction. In G. I. Bell & T. G. Marr (Eds.), *Computers and DNA*. Redwood City, CA: Addison-Wesley.

Blickle, T., & Thiele, L. (1994). Genetic programming and redundancy. In *Genetic algorithms within the framework of evolutionary computation (Workshop at the Eighteenth Annual German Conference on Artificial Intelligence)*, (pp. 33–38).

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). *Artificial intelligence through simulated evolution*. New York, NY: John Wiley & Sons.

Forrest, S., & Mitchell, M. (1992). Relative building-block fitness and the building-block hypothesis. In D. Whitley (Ed.), *Foundations of genetic algorithms 2* (pp. 109–126). San Mateo, CA: Morgan Kaufmann.

Gilbert, W. (1987). The exon theory of genes. *Cold Spring Harbor Symposia on Quantitative Biology, 52*, 901–905.

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems, 3*, 493–530.

Harvey, I. (1992). Species adaptation genetic algorithms: A basis for a continuing SAGA. In F. J. Varela & P. Bourgine (Eds.), *Proceedings of the First European Conference on Artificial Life* (pp. 346–554). Cambridge, MA: MIT Press.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Koza, J. (1992). *Genetic programming*. Cambridge, MA: MIT Press.

Langdon, W. B., & Poli, R. (1997). Fitness causes bloat. In *Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing*. http://www.bath.ac.uk/Departments/Eng/wsc2/home.html

Levenick, J. R. (1991). Inserting introns improves genetic algorithm success rate: Taking a cue from biology. In R. K. Belew & L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms* (pp. 123–127). San Mateo, CA: Morgan Kaufmann.

Lewin, B. (1994). *Genes 5*. New York, NY: Oxford University Press.

Nei, M. (1987). *Molecular evolutionary genetics*. New York, NY: Columbia University Press.

Nordin, P., & Banzhaf, W. (1995). Complexity compression and evolution. In L. Eshelman (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* (pp. 310–317). San Mateo, CA: Morgan Kaufmann.

Nordin, P., Francone, F., & Banzhaf, W. (1996). Explicitly defined introns and destructive crossover in genetic programming. In K. E. Kinnear (Ed.), *Advances in genetic programming 2* (pp. 111–134). Cambridge, MA: MIT Press.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Frommann-Holzboog.

Rosca, J. (1996). Generality versus size in genetic programming. In J. R. Koza, D. E. Goldberg, D. B. Fogel, & R. L. Riolo (Eds.), *Genetic programming 1996*. Cambridge, MA: MIT Press.

Soule, T., Foster, J. A., & Dickinson, J. (1996). Code growth in genetic programming. In J. R. Koza, D. E. Goldberg, D. B. Fogel, & R. L. Riolo (Eds.), *Genetic programming 1996* (pp. 215–233). Cambridge, MA: MIT Press.

Wu, A. S., & Lindsay, R. K. (1995). Empirical studies of the genetic algorithm with noncoding segments. *Evolutionary Computation, 3*(2), 121–147.

Wu, A. S., & Lindsay, R. K. (1996a). A comparison of the fixed and floating building-block representation in the genetic algorithm. *Evolutionary Computation, 4*(2), 169–193.

Wu, A. S., & Lindsay, R. K. (1996b). A survey of intron research in genetics. In H.-M. Voight, W. Ebeling, I. Rechenberg, & H.-P. Schwefel (Eds.), *Proceedings of the Fourth International Conference on Parallel Problem Solving from Nature* (pp. 101–110). Berlin: Springer-Verlag.