Editorial Introduction

Welcome to the second issue of volume 3 of *Evolutionary Computation*! By the time you are reading this, we will be in the middle of summer conference activity. A quick glance at the Calendar of Events section will convince you of the enormous interest in and activities involving evolutionary computation. Trying to stay current with one's own field, much less related ones, is becoming an increasingly difficult task. One of the things that is very helpful in this respect is to publish summaries of the EC-related activities at recent conferences and workshops. Now all we need are volunteers.

One of the issues of continuing importance in evolutionary computation is the assessment of various features of biological evolution with respect to their computational usefulness. An example of this is the well-known fact that much of the DNA in natural organisms is noncoding DNA, that is, it does not directly code for the RNA required for protein synthesis. So, from a genetic coding point of view, this looks like a representation that deliberately includes excess bits of information. There has been considerable speculation as to why such a representation occurs in natural systems, the most common of which suggest that redundancy, internal workspace areas, and spacing on the genome play important roles in an evolutionary process that proceeds via the processes of recombination and mutation. A variety of work has explored the computational usefulness of such representations for evolutionary algorithms with mixed results.

The first article in this issue continues to explore this issue. Wu and Lindsay articulate a number of possible hypotheses relating to the introduction of noncoding segments into the binary string representation of a traditional GA, and test these hypotheses on some of the Royal Road functions. They find little evidence that noncoding segments help on simple problems, but suggest that such representations provide a stabilizing effect that helps on more complex problems.

An increasing amount of attention is being given to evolutionary algorithms designed to evolve objects that are considerably more complex than a fixed set of parameter values. The most ambitious of these involve the evolution of programs for controlling the behavior of an artificial agent interacting with its environment. An interesting question arises as to the form such control programs should take. Holland proposed a "classifier system" model in which simple situation-action rules both cooperated and competed with each other for "payoff" flowing back from the environment. This model has been extensively studied and is the subject of two articles in this issue.

The first, by Wilson, extends the standard classifier model in two significant directions: by using prediction accuracy as a measure of rule fitness, and by introducing a nonrandom mating scheme based on match sets that produces a useful niching effect. He tests these ideas in a variety of contexts including multiplexer and 2-D animate problems, and reports significant improvement in both the accuracy and the generality of the rules produced.

The second classifier article, by Gilbert, Bell, and Valenzuela, provides a nice picture of the issues involved in applying classifier system models to real-world control problems. They describe a difficult, large-scale adaptive control problem involving a chemical batch reaction in which profit optimization is the objective function. In order to achieve their goals, they found it necessary to modify the classical model in several respects, including the use of a profit-sharing plan and allowing the specification of multiple actions. With these changes they were able to demonstrate significant learning in an off-line simulated environment. They note and discuss several remaining issues that need to be addressed before one might use their system for on-line control of a real chemical plant.

An alternate strategy for evolving control programs is to express them in a more standard programming language. Koza and others have extensively studied the merits of using LISP as the representation language, and have reported positive results for a wide range of problems. The article by Montana in this issue contributes to this approach by studying the effects of introducing strong data typing into the semantics of the LISP code being evolved. He reports the results of a number of experiments that show an improvement in both the learning time and the generality of the solutions found.

Volume 3 continues to take shape with a variety of interesting articles in preparation. I'll be posting the contents of upcoming issues on MIT Press' WWW server, so get in the habit of checking out what's new in the ECJ section.

Enjoy!

Kenneth De Jong
Computer Science Department
George Mason University
Fairfax, VA 22030 USA
ecj@cs.gmu.edu