An Overview of Evolutionary Computation

by
William M. Spears
Kenneth A. De Jong
Thomas Baeck
David B. Fogel
Hugo de Garis

This paper was published in the proceedings of the 1993 European Conference on Machine Learning.

Abstract

Evolutionary computation uses computational models of evolutionary processes as key elements in the design and implementation of computer-based problem solving systems. In this paper we provide an overview of evolutionary computation, and describe several evolutionary algorithms that are currently of interest. Important similarities and differences are noted, which lead to a discussion of important issues that need to be resolved, and items for future research.

1. Introduction

Evolutionary computation uses computational models of evolutionary processes as key elements in the design and implementation of computer-based problem solving systems. There are a variety of evolutionary computational models that have been proposed and studied which we will refer to as evolutionary algorithms. They share a common conceptual base of simulating the evolution of individual structures via processes of selection and reproduction. These processes depend on the perceived performance (fitness) of the individual structures as defined by an environment.

More precisely, evolutionary algorithms maintain a population of structures that evolve according to rules of selection and other operators, such as recombination and mutation. Each individual in the population receives a measure of its fitness in the environment. Selection focuses attention on high fitness individuals, thus exploiting the available fitness information. Recombination and mutation perturb those individuals, providing general heuristics for exploration. Although simplistic from a biologist's viewpoint, these algorithms are sufficiently complex to provide robust and powerful adaptive search mechanisms.

Figure 1 outlines a typical evolutionary algorithm (EA). A population of individual structures is initialized and then evolved from generation to generation by repeated applications of evaluation, selection, recombination, and mutation. The population size N is generally constant in an evolutionary algorithm, although there is no a priori reason (other than convenience) to make this assumption. We will discuss the issue of a dynamic population size later in this paper.

```
procedure EA; {
t = 0;
initialize population P(t);
evaluate P(t);
until (done) {
        t = t + 1;
        parent_selection P(t);
        recombine P(t);
        mutate P(t);
        evaluate P(t);
        survive P(t);
}       }
```

Fig. 1. A typical evolutionary algorithm

An evolutionary algorithm typically initializes its population randomly, although domain specific knowledge can also be used to bias the search. Evaluation measures the fitness of each individual according to its worth in some environment. Evaluation may be as simple as computing a fitness function or as complex as running an elaborate simulation. Selection is often performed in two steps, parent selection and survival. Parent selection decides who becomes parents and how many children the parents have. Children are created via recombination, which exchanges information between parents, and mutation, which further perturbs the children. The children are then evaluated. Finally, the survival step decides who survives in the population.

Let us illustrate an evolutionary algorithm with a simple example. Suppose an automotive manufacturer wishes to design a new engine and fuel system in order to maximize performance, reliability, and gas-mileage, while minimizing emissions. Let us further suppose that an engine simulation unit can test various engines and return a single value indicating the fitness score of the engine. However, the number of possible engines is large and there is insufficient time to test them all. How would one attack such a problem with an evolutionary algorithm?

First, we define each individual to represent a specific engine. For example, suppose the cubic inch displacement (CID), fuel system, number of valves, cylinders, and presence of turbo-charging are all engine variables. The initialization step would create an initial population of possible engines. For the sake of simplicity, let us assume a (very small) population of size four. Here is an example initial population:

Individual	CID	Fuel System	Turbo	Valves	Cylinders
1	350	4 Barrels	Yes	16	8
2	250	Mech. Inject.	No	12	6
3	150	Elect. Inject.	Yes	12	4
4	200	2 Barrels	No	8	4

We now evaluate each individual with the engine simulator. Each individual receives a fitness score (the higher the better):

Individual	CID	Fuel System	Turbo	Valves	Cylinders	Score
1	350	4 Barrels	Yes	16	8	50
2	250	Mech. Inject.	No	12	6	100
3	150	Elect. Inject.	Yes	12	4	300
4	200	2 Barrels	No	8	4	150

Parent selection decides who has children and how many to have. For example, we could decide that individual 3 deserves two children, because it is so much better than the other individuals. Children are created through recombination and mutation. As mentioned above, recombination exchanges information between individuals, while mutation perturbs individuals, thereby increasing diversity. For example, recombination of individuals 3 and 4 could produce the two children:

Individual	CID	Fuel System	Turbo	Valves	Cylinders
3'	200	Elect. Inject.	Yes	8	4
4'	150	2 Barrels	No	12	4

Note that the children are composed of elements of the two parents. Further note that the number of cylinders must be four, because individuals 3 and 4 both had four cylinders. Mutation might further perturb these children, yielding:

Individual	CID	Fuel System	Turbo	Valves	Cylinders
3'	250	Elect. Inject.	Yes	8	4
4'	150	2 Barrels	No	12	6

We now evaluate the children, giving perhaps:

Individual	CID	Fuel System	Turbo	Valves	Cylinders	Score
3'	250	Elect. Inject.	Yes	8	4	250
4'	150	2 Barrels	No	12	6	350

Finally we decide who will survive. In our constant population size example, which is typical of most EAs, we need to select four individuals to survive. How this is accomplished varies considerably in different EAs. If, for example, only the best individuals survive, our population would become:

Individual	CID	Fuel System	Turbo	Valves	Cylinders	Score
3	150	Elect. Inject.	Yes	12	4	300
4	200	2 Barrels	No	8	4	150
3'	250	Elect. Inject.	Yes	8	4	250
4'	150	2 Barrels	No	12	6	350

This cycle of evaluation, selection, recombination, mutation, and survival continues until some termination criterion is met.

This simple example serves to illustrate the flavor of an evolutionary algorithm. It is important to point out that although the basic conceptual framework of all EAs is similar, their particular implementations differ in many details. For example, there are a wide variety of selection mechanisms. The representations of individuals ranges from bit-strings to real-valued vectors, Lisp expressions, and neural networks. Finally, the relative importance of mutation and crossover (recombination), as well as their particular implementations, differs widely across evolutionary algorithms.

The remainder of this paper is organized into three sections. First, we will continue our introduction to evolutionary algorithms by describing at a high level a variety of implementations. Second, we will discuss the issues underlying the differences between the implementations, taking the opportunity to provide comparisons at a finer level of detail. Finally we will discuss how these issues might be resolved and summarize recent work in this area. Our goal is to encourage increased discussion, with the eventual hope for more powerful and robust evolutionary algorithms.

2. Varieties of Evolutionary Algorithms

The origins of evolutionary algorithms can be traced to at least the 1950's (e.g., Fraser, 1957; Box, 1957). For the sake of brevity we will not concentrate on this early work but will discuss in some detail three methodologies that have emerged in the last few decades: "evolutionary programming" (Fogel et al., 1966), "evolution strategies" (Rechenberg, 1973), and "genetic algorithms" (Holland, 1975).

Although similar at the highest level, each of these varieties implements an evolutionary algorithm in a different manner. The differences touch upon almost all aspects of evolutionary algorithms, including the choices of representation for the individual structures, types of selection mechanism used, forms of genetic operators, and measures of performance. We will highlight the important differences (and similarities) in the following sections, by examining some of the variety represented by the current family of evolutionary algorithms.

These approaches in turn have inspired the development of additional evolutionary algorithms such as "classifier systems" (Holland, 1986), the LS systems (Smith, 1983), "adaptive operator" systems (Davis, 1989), GENITOR (Whitley, 1989), SAMUEL (Grefenstette, 1989), "genetic programming" (de Garis, 1990; Koza, 1991), "messy GAs" (Goldberg, 1991), and the CHC approach (Eshelman, 1991). We will not attempt to survey this broad range of activities here. The interested reader is encouraged to peruse the recent literature for more details (e.g., Belew and Booker, 1991; Fogel and Atmar, 1992; Whitley, 1992; Manner and Manderick, 1992).

2.1 Evolutionary Programming

Evolutionary programming (EP), developed by Fogel et al. (1966) traditionally has used representations that are tailored to the problem domain. For example, in real-valued optimization problems, the individuals within the population are real-valued vectors. Similarly, ordered lists are used for traveling salesman problems, and graphs for applications with finite state machines. EP is often used as an optimizer, although it arose from the desire to generate machine intelligence.

The outline of the evolutionary programming algorithm is shown in Figure 2.

```
procedure EP; {
t = 0;
initialize population P(t);
```

```
evaluate P(t);
until (done) {
        t = t + 1;
        parent_selection P(t);
        mutate P(t);
        evaluate P(t);
        survive P(t);
}       }
```

Fig. 2. The evolutionary programming algorithm

After initialization, all N individuals are selected to be parents, and then are mutated, producing N children. These children are evaluated and N survivors are chosen from the 2N individuals, using a probabilistic function based on fitness. In other words, individuals with a greater fitness have a higher chance of survival. The form of mutation is based on the representation used, and is often adaptive (see Section 3.2). For example, when using a real-valued vector, each variable within an individual may have an adaptive mutation rate that is normally distributed with a zero expectation. Recombination is not generally performed since the forms of mutation used are quite flexible and can produce perturbations similar to recombination, if desired. As discussed in a later section, one of the interesting and open issues is the extent to which an EA is affected by its choice of the operators used to produce variability and novelty in evolving populations.

2.2 Evolution Strategies

Evolution strategies (ESs) were independently developed by Rechenberg (1973), with selection, mutation, and a population of size one. Schwefel (1981) introduced recombination and populations with more than one individual, and provided a nice comparison of ESs with more traditional optimization techniques. Due to initial interest in hydrodynamic optimization problems, evolution strategies typically use real-valued vector representations.

Figure 3 outlines a typical evolution strategy (ES). After initialization and evaluation, individuals are selected uniformly

```
procedure ES; {
t = 0;
initialize population P(t);
evaluate P(t);
until (done) {
        t = t + 1;
        parent_selection P(t);
        recombine P(t)
        mutate P(t);
        evaluate P(t);
        survive P(t);
}       }
```

Fig. 3. The evolution strategy algorithm

randomly to be parents. In the standard recombinative ES, pairs of parents produces children via recombination, which are further perturbed via mutation. The number of children created is greater than N. Survival is deterministic and is implemented in one of two ways. The first allows the N best children to survive, and replaces the parents with these children. The second allows the N best children and parents to survive. Like EP, considerable effort has focused on adapting mutation as the algorithm runs by allowing each variable within an individual to have an adaptive mutation rate that is normally distributed with a zero expectation. Unlike EP, however, recombination does play an important role in evolution strategies, especially in adapting mutation.

2.3 Genetic Algorithms

Genetic algorithms (GAs), developed by Holland (1975), have traditionally used a more domain independent representation, namely, bit-strings. However, many recent applications of GAs have focused on other representations, such as graphs (neural networks), Lisp expressions, ordered lists, and real-valued vectors.

Figure 4 outlines a typical genetic algorithm (GA).

```
procedure GA; {
t = 0;
initialize population P(t);
evaluate P(t);
until (done) {
        t = t + 1;
        parent_selection P(t);
        recombine P(t)
        mutate P(t);
        evaluate P(t);
        survive P(t);
}       }
```

Fig. 4. The genetic algorithm

After initialization parents are selected according to a probabilistic function based on relative fitness. In other words, those individuals with higher relative fitness are more likely to be selected as parents. N children are created via recombination from the N parents. The N children are mutated and survive, replacing the N parents in the population. It is interesting to note that the relative emphasis on mutation and crossover is opposite to that in EP. In a GA mutation flips bits with some small probability, and is often considered to be a background operator. Recombination, on the other hand, is emphasized as the primary search operator. GAs are often used as optimizers, although some researchers emphasize its general adaptive capabilities (De Jong, 1992).

2.4 Variations on these Themes

These three approaches (EP, ESs, and GAs) have served to inspire an increasing amount of research on and development of new forms of evolutionary algorithms for use in specific problem solving contexts. A few of these are briefly described below, selected primarily to give the reader a sense of the variety of directions being explored.

One of the most active areas of application of evolutionary algorithms is in solving complex function and combinatorial optimization problems. A variety of features are typically added to EAs in this context to improve both the speed and the precision of the results. Interested readers should review Davis' work on real-valued representations and adaptive operators (Davis, 1989), Whitley's GENITOR system incorporating ranking and "steady state" mechanisms (Whitley, 1989), Goldberg's "messy GAs", that involve adaptive representations (Goldberg, 1991), and Eshelman's high-powered CHC algorithm (Eshelman, 1991).

A second active area of application of EAs is in the design of robust rule learning systems. Holland's (1986) classifier systems were some of the early examples, followed by the LS systems of Smith (1983). More recent examples include the SAMUEL system developed by Grefenstette (1989), the GABIL system of De Jong and Spears (1991), and the GIL system of Janikow (1991). In each case, significant adaptations to the basic EAs have been made in order to effectively represent, evaluate, and evolve appropriate rule sets as defined by the environment.

One of the most fascinating recent developments is the use of EAs to evolve more complex structures such as neural networks and Lisp code. This has been dubbed "genetic programming", and is exemplified by the work of de Garis (1990), Fujiko and Dickinson (1987), and Koza (1991). de Garis evolves weights in neural networks, in an attempt to build complex behavior. Fujiko and Dickinson evolved Lisp expressions to solve the Prisoner's Dilemma. Koza also represents individuals using Lisp expressions and has solved a large number of optimization and machine learning tasks. One of the open questions here is precisely what changes to EAs need to be made in order to efficiently evolve such complex structures.

3. Issues

In the previous section we highlighted the similarities and differences of the various forms of evolutionary algorithms. The differences arise from a number of relevant issues. In this section we will explore these issues briefly, and take the opportunity to also define the algorithms above in greater detail.

3.1 Scaling, Selection and Fitness