An Overview of Evolutionary Computation †

William M. Spears ‡
Kenneth A. De Jong
Thomas Bäck
David B. Fogel
Hugo de Garis

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

---

† The paper will appear in the proceedings of the 1993 European Conference on Machine Learning.

‡ The first author is affiliated with the AI Center of the Naval Research Laboratory in Washington, DC 20375 USA. His email address is spears@aic.nrl.navy.mil

```
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

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.

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; Männer 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. After initialization, all *N* individuals are selected to be parents, and then are mutated,

```
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

producing *N* children. These children are evaluated and *N* survivors are chosen from the 2*N* 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 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

```
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

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). 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).

```
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

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

Central to every evolutionary algorithm is the concept of fitness (i.e., evaluation). If we assume, without loss of generality, that we wish to maximize fitness, then we wish to concentrate search in those areas of higher fitness. This concentration of effort, commonly referred to by the term *exploitation*, is the task of selection. Each EA addresses this issue in a different manner.

Before we describe selection mechanisms further, it is also important to consider the issue of *scaling*. Suppose one has two search spaces. The first is described with a real-valued fitness function F. The second search space is described by a fitness function G that is equivalent to F^p, where p is some constant. The relative *positions* of peaks and valleys in the two search spaces correspond exactly. Only the relative *heights* differ (i.e., the vertical scale is different). Should our EA search both spaces in the same manner?

There is no right or wrong answer to this question, since it really depends on our goals and the problems to be solved. If we believe that the EA should search the two spaces in the same manner, then selection should only be based on the relative ordering of fitnesses. ESs, for example, use precisely this method. Parent selection is performed uniformly randomly, with no regard to fitness. Survival simply saves the N best individuals, which is only based on the relative ordering of fitnesses. This form of selection is often referred to as *ranking* selection, since only the rank of individuals is of importance. EP selection is similar to that of the ES algorithm. All individuals are selected to be parents. Each parent is mutated once, producing N children. A probabilistic ranking mechanism chooses the N best individuals for survival, from the union of the parents and children. Again, this is a selection mechanism based on rank.

The GA community has also advocated ranking for some situations, but by and large many members believe that F and G should be searched differently. Fitness proportional selection is the probabilistic selection mechanism of the traditional GA. Parent selection is performed based on how fit an individual is with respect to the population average. For example, an individual with fitness twice the population average will tend to have twice as many children as average individuals. Survival, though, is not based on fitness, since the parents are automatically replaced by the children.

One problem with this latter approach is that, as the search continues, more and more individuals receive fitnesses with small relative differences. This lessens the selection pressure, slowing the progress of the search. This effect, often referred to as "lacking the killer instinct", can be compensated somewhat by scaling mechanisms, that attempt to magnify relative differences as the search progresses. A number of scaling mechanisms exists, but their description is beyond the scope of this paper. The interested reader is urged to refer to Grefenstette and Baker (1989) for an investigation into the relationships between fitness, scaling, and selection.

3.2 Mutation and Adaptation

As mentioned earlier, selection serves to focus search into areas of high fitness. Of course, if selection were the only genetic operator, the population would never have any individuals other than those introduced in the initial population. Other genetic operators perturb these individuals, providing exploration in nearby areas. Although a number of operators are possible, we will concentrate on the two predominant operators, namely, mutation and recombination.

The importance of mutation in EAs varies widely. Koza (1991) does not use mutation at all. GAs typically use mutation as a simple background operator, to ensure that a particular bit value is not lost forever. Using our previous example, suppose every member of our engine population had four cylinders. Then mutation can reintroduce six and eight cylinder engines. Recall that GAs traditionally work on bit-strings. Under these conditions, mutation in GAs typically flips bits with a very low probability (e.g., 1 bit out of 1000).

Mutation is far more important in ESs and EP. Instead of a global mutation rate, mutation probability distributions can be maintained for every variable of every individual. Thus, each variable can be mutated according to a different probability distribution. More importantly, ESs and EP encode the probability distributions as extra information within each individual, and allow this information to evolve as well. What is achieved is the *self-adaptation* of mutation parameters, *while* the space is being searched. Again, full details of this are beyond the scope of this paper. The interested reader is encouraged to read Bäck et al. (1991), Bäck and Schwefel (1993), and Fogel (1992).

3.3 Recombination and Adaptation

Recombination is the other predominant genetic operator. Recombination merges variables from two parents to produce offspring that have variables from both parents. Like mutation, the relative importance of recombination in various EAs varies widely. EP does not make use of recombination. Koza (1991) only uses recombination to form new Lisp expressions.

ESs and GAs use both recombination and mutation. There are a number of recombination methods for ESs, all of which assume that the individuals are composed of real-valued variables. Either the values are exchanged (as in our "engine" example above), or they are averaged. For example, a four cylinder parent could recombine with an eight cylinder parent to produce a six cylinder child. Finally, the ES community has also considered multi-parent versions of these operators. Although the ES community places more emphasis on mutation, and does not adaptively modify crossover, they also feel crossover is essential for the proper adaptation of the mutation parameters.

The GA community places primary emphasis on crossover, and a number of recombination operators are widely used. Again, for the sake of brevity, we will only discuss the most popular, namely, *one-point*, *multi-point* and *uniform* recombination. One-point recombination inserts a *cut-point* within the two parents (e.g., between the 3rd and 4th variables, or bits). Then the information before the cut-point is swapped between the two parents. Multi-point recombination is a generalization of this idea, introducing a higher number of cut-points. Information is then swapped between pairs of cut-points. Uniform crossover, however, does not use cut-points, but simply uses a global parameter to indicate the likelihood that each variable should be exchanged between two parents. Considerable experimental and theoretical work has investigated the differences between these forms of recombination. Spears and De Jong (1992), Booker (1992), and Vose and Liepins (1991) provide theoretical comparisons.

Despite the emphasis on recombination within the GA community, interest in mutation has increased recently, partly due to the influence of the ES and EP communities. Schaffer and Eshelman (1991) have experimentally shown that mutation is a powerful search operator in its own right, while still maintaining the usefulness of crossover in certain situations. Spears (1992a) agrees with this view, and has theoretically shown some of the strengths and weakness of mutation and recombination. It is important to realize that recombination and mutation provide different search biases, which may or may not be appropriate for the task at hand. Since a priori appropriateness may be hard to determine, the key to more robust EA systems probably lies in the adaptive selection of such genetic operators. Unfortunately, very little has been done in the way of adaptive recombination. Schaffer and Morishima (1987) have experimented with *punctuation-marks* that indicate where good cut-points may exist. Davis (1989) has experimented with adapting the rate at which recombination is applied, given performance feedback. Finally, Spears (1992b) has shown that it is feasible for the GA to choose between two forms of recombination. Clearly, however, this is an area for further research.

3.4 Representation

Of course, any genetic operator such as mutation and recombination must be defined with a particular individual representation in mind. Again, the EA community differs widely in the representations used. Traditionally, GAs use bit strings. In theory, this representation makes the GA more problem independent, because once a bit string representation is found, standard bit-level mutation and recombination can often be used. We can also see this as a more genotypic level of representation, since the individual is in some sense encoded in the bit string. Recently, however, the GA community has investigated more phenotypic representations, including vectors of real values (Davis, 1989), ordered lists (Whitley et al., 1989), neural networks (Harp et. al, 1991), and Lisp expressions (Koza, 1991). For each of these representations, special mutation and recombination operators are introduced. The EP and ES communities are similar in this regard. The ES and EP communities focus on real-valued vector representations, although the EP community has also used ordered list and finite state automata representations, as suggested by the domain of the problem.

Although much has been done experimentally, very little has been said theoretically that helps one choose good representations, nor that explains what it *means* to have a good representation. Also, very little has been done in the way of adaptive representations, with the exception of messy GAs (Goldberg, 1991), Argot (Shaefer, 1987), the dynamic parameter encoding (DPE) scheme of Schraudolph and Belew

(1992), and the Delta coding of Whitley et al. (1991). Messy GAs, Argot, DPE, and Delta coding all attempt to manipulate the granularity of the representation, thus focusing search at the appropriate level. Despite some initial success in this area, it is clear that much more work needs to be done.

3.5 Adaptive EAs

Despite some work on adapting representation, mutation, and recombination within evolutionary algorithms, very little has been accomplished with respect to the adaptation of population sizes and selection mechanisms. One way to characterize selection is by the *strength* of the selection mechanism. Strong selection refers to a selection mechanism that concentrates quickly on the best individuals, while weaker selection mechanisms allow poor individuals to survive (and produce children) for a longer period of time. Similarly, the population can be thought of as having a certain *carrying capacity*, which refers to the amount of information that the population can usefully maintain. A small population has less carrying capacity, which is usually adequate for simple problems. Larger populations, with larger carrying capacities, are often better for more difficult problems. Although some work has attempted to characterize good population sizes (Goldberg, 1989a), more theory is needed. In lieu of theory, then, perhaps the evolutionary algorithm can adapt both selection pressure and the population size dynamically, as it solves problems.

3.6 Performance Measures, EA-Hardness, and Evolvability

Of course, one can not refer to adaptation without having a performance goal in mind. EP and ES usually have optimization for a goal. In other words, they are typically most interested in finding the best solution as quickly as possible. The GA community has often taken a similar stance, although there is also some concern that such a stance can be somewhat misleading. De Jong (1992) reminds us that GAs are *not* function optimizers per se, although they can be used as such. There is very little theory indicating how well GAs will perform optimization tasks. Instead, theory concentrates on what is referred to as *accumulated payoff.* The difference can be illustrated by considering financial investment planning over a period of time (e.g., you play the stock market). Instead of trying to find the *best* stock, you are trying to maximize your returns as the various stocks are sampled. Clearly the two goals are somewhat different, and maximizing the return may or may not also be a good heuristic for finding the best stock. This difference in emphasis clearly has implications for how an EA practitioner can (and should) measure performance, which will have further implications for how adaptation should be accomplished.

This difference also colors much of the discussion concerning the issue of problem difficulty. The GA community refers to hard problems as *GA-Hard*. Since we are now in the broader context of EAs, let us refer to hard problems as *EA-Hard*. Often, a problem is considered difficult if the EA can not find the optimum. Although this is a quite reasonable definition, difficult problems are often constructed by taking advantage of the EA in such a way that selection deliberately leads the search away from the optimum. Such problems are called *deceptive* (Goldberg, 1989b). From a function optimization point of view, the problem is indeed deceptive. However, the EA may nonetheless maximize accumulated payoff. Should we call a deceptive problem EA-Hard? The answer obviously depends on our goals.

It is clear, then, that problem difficulty is a function of the problem, the goal, and the algorithm used to solve that problem. Although deceptiveness is one possibility, other measures of problem difficulty are needed.† One possibility is *fitness correlation*, which appears to be a measure of EA-Hardness that places less emphasis on optimality (Manderick et al., 1991). Fitness correlation measures the correlation between the fitness of children and their parents. Manderick et al. found a strong relationship between GA performance and the strength of the correlations. Similarly, Lipsitch (1991) has also proposed examining fitness correlations. Another possibility is problem modality. Those problems that have many suboptimal solutions will, in general, be more difficult to search. Finally, this issue is also very related to a concern of de Garis, which he refers to as *evolvability*. de Garis notes that often his systems do not evolve at all, namely, that fitness does not increase over time. The reasons for this are not clear and remain an important research topic.

3.7 Distributed EAs

Because of the inherent natural parallelism within an EA, much recent work has concentrated on the implementation of EAs on parallel machines. Typically either one processor holds one individual (in SIMD machines), or a subpopulation (in MIMD machines). Clearly, such implementations hold promise of execution time decreases. More interestingly, though, for the topic of this paper, are the evolutionary effects that can be naturally illustrated with parallel machines, namely, speciation, nicheing, and punctuated equilibria. Belew and Booker (1991) contain many examples of the most current work in this area.

4 Current Trends

With a better understanding of the similarities and differences between various implementations of EAs, the community has begun to concentrate on generalizing results initially shown only for specific EAs. For example, Grefenstette and Baker (1989) illustrate that many features of EAs do not change when certain properties of selection and scaling are assumed. They also indicate when the features change, if the properties are not met. Although this work is preliminary, it helps explain why a wide variety of EAs have all met with success. Bäck is also investigating the differences between GAs and ESs and is attempting to merge the best features of each in order to have a more robust EA.

---

† For an analysis of deception, see Grefenstette (1992).

As we understand better the strengths and weaknesses of our current evolutionary models, it is also important to revisit the biological and evolutionary literature for new insights and inspirations for enhancements. Booker (1992) has recently pointed out the connections with GA recombination theory to the more general theory of population genetics recombination distributions. Mühlenbein (1993) has concentrated on EAs that are modeled after breeding practices. In the EP community, Atmar (1992) highlights some errors common to evolutionary theory and the EA community.

5 Summary

We have attempted to provide a brief overview of the field of evolutionary computation by describing three classes of evolutionary algorithms which have served to define and shape the field. By highlighting their similarities and differences, we have identified a number of important issues that suggest directions for future research.

With the rapid growth of the field, there is a particularly pressing need to extend existing and developing new analysis tools which allow us to better understand and evaluate the emerging varieties of EAs and their applications. We hope this paper will serve as a catalyst for such activities.

Affiliations

William M. Spears is affiliated with the Naval Research Laboratory (USA), Kenneth A. De Jong with George Mason University (USA), Thomas Bäck with the University of Dortmund (Germany), David B. Fogel with ORINCON Corporation (USA), and Hugo de Garis with the ATR Laboratory (Japan).

References

Atmar, W. (1992) The philosophical errors that plague both evolutionary theory and simulated evolutionary programming. *Proceedings of the First Annual Conference on Evolutionary Programming*, 27-34. San Diego, CA: Evolutionary Programming Society.

Bäck, T., Hoffmeister, F., & Schwefel, H.-P. (1991) A survey of evolution strategies. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 2-9. La Jolla, CA: Morgan Kaufmann.

Bäck, T., & Schwefel, H.-P. (1993) An overview of evolutionary algorithms for parameter optimization. Submitted to the *Journal of Evolutionary Computation*.

Belew, R. K., & Booker, L. B. (eds.) (1991) *Proceedings of the Fourth International Conference on Genetic Algorithms*. La Jolla, CA: Morgan Kaufmann.

Booker, L. B. (1992) Recombination distributions for genetic algorithms. *Proceedings of the Foundations of Genetic Algorithms Workshop*. Vail, CO: Morgan Kaufmann.

Box, G. E. P. (1957) Evolutionary operation: a method of increasing industrial productivity. *Applied Statistics*, Vol. 6, 81-101.

Davis, L. (1989) Adapting operator probabilities in genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, 60-69. La Jolla, CA: Morgan Kaufmann.

de Garis, H. (1990) Genetic programming: modular evolution for darwin machines. *Proceedings of the 1990 International Joint Conference on Neural Networks*, 194-197. Washington, DC: Lawrence Erlbaum.

De Jong, K. A. (1975) *An analysis of the behavior of a class of genetic adaptive systems.* Doctoral Thesis, Department of Computer and Communication Sciences. University of Michigan, Ann Arbor.

De Jong, K. & Spears, W. (1991) Learning concept classification rules using genetic algorithms. *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, 651-656. Sydney, Australia: Morgan Kaufmann.

De Jong, K. A. (1992) Are genetic algorithms function optimizers? *Proceedings of the Second International Conference on Parallel Problem Solving from Nature*.

Eshelman, L. J., & Schaffer, J. D. (1991) Preventing premature convergence in genetic algorithms by preventing incest. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 115-122. La Jolla, CA: Morgan Kaufmann.

Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966) *Artificial Intelligence Through Simulated Evolution.* New York: Wiley Publishing.

Fogel, D. B. (1992) An analysis of evolutionary programming. *Proceedings of the First Annual Conference on Evolutionary Programming*, 43-51. La Jolla, CA: Evolutionary Programming Society.

Fogel, D. B., & Atmar, J. W. (eds.) (1992) *Proceedings of the First Annual Conference on Evolutionary Programming*. La Jolla, CA: Evolutionary Programming Society

Fraser, A. S. (1957) Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Science*, 10, 484-491.

Fujiko, C., & Dickinson, J. (1987) Using the genetic algorithm to generate LISP source code to solve the prisoner's dilemma. *Proceedings of the Second International Conference on Genetic Algorithms*, 236-240. Cambridge, MA: Lawrence Erlbaum.

Goldberg, D. E. (1989a) Sizing populations for serial and parallel genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, 70-79. Fairfax, VA: Morgan Kaufmann.

Goldberg, D. E. (1989b) *Genetic Algorithms in Search, Optimization & Machine Learning*. Reading, MA: Addison-Wesley.

Goldberg, D. E., Deb, K., & Korb, B. (1991) Don't worry, be messy. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 24-30. La Jolla, CA: Morgan Kaufmann.

Grefenstette, J. G., and Baker, J. E. (1989) How genetic algorithms work: a critical look at implicit parallelism. *Proceedings of the Third International Conference on Genetic Algorithms*, 20-27. Fairfax, VA: Morgan Kaufmann.

Grefenstette, John J. (1989) A system for learning control strategies with genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, 183-190. Fairfax, VA: Morgan Kaufmann.

Grefenstette, J. G. (1992) Deception considered harmful. *Proceedings of the Foundations of Genetic Algorithms Workshop*. Vail, CO: Morgan Kaufmann.

Harp, S. A., Samad, T., & Guha, A. (1991) Towards the genetic synthesis of neural networks. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 360-369. La Jolla, CA: Morgan Kaufmann.

Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems.* Ann Arbor, Michigan: The University of Michigan Press.

Holland, J. (1986) Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. Michalski, J. Carbonell, T. Mitchell (eds.), *Machine Learning: An Artificial Intelligence Approach.* Los Altos: Morgan Kaufmann.

Janikow, C. (1991) *Inductive learning of decision rules from attribute-based examples: A knowledge-intensive genetic algorithm approach*. TR91-030, The University of North Carolina at Chapel Hill, Dept. of Computer Science, Chapel Hill, NC.

Koza, J. R. (1991) Evolving a computer program to generate random numbers using the genetic programming paradigm. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 37-44. La Jolla, CA: Morgan Kaufmann.

Lipsitch, M. (1991) Adaptation on rugged landscapes generated by iterated local interactions of neighboring genes. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 128-135. La Jolla, CA: Morgan Kaufmann.

Manderick, B., de Weger, M., & Spiessens, P. (1991) The genetic algorithm and the structure of the fitness landscape. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 143-149. La Jolla, CA: Morgan Kaufmann.

Männer, R., & Manderick, B. (1992) *Proceedings of the Second International Conference on Parallel Problem Solving from Nature*, Amsterdam: North Holland.

Mühlenbein, H., & Schlierkamp-Voosen, D. (1993) The distributed breeder genetic algorithm. Submitted to the *Journal of Evolutionary Computation*.

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

Schaffer, J. D., Eshelman, L. J. (1991) On crossover as an evolutionarily viable strategy. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 61-68. La Jolla, CA: Morgan Kaufmann.

Schaffer, J. D. & Morishima, A. (1987) An adaptive crossover distribution mechanisms for genetic algorithms. *Proceedings of the Second International Conference on Genetic Algorithms*, 36-40. Cambridge, MA: Lawrence Erlbaum.

Schraudolph, N. N., & Belew, R. K. (1992) Dynamic parameter encoding for genetic algorithms. *Machine Learning Journal*, Volume 9, Number 1, 9-22.

Schwefel, H.-P. (1981) *Numerical Optimization of Computer Models*. New York: John Wiley & Sons.

Shaefer, C. G. (1987) The ARGOT strategy: adaptive representation genetic optimizer technique. *Proceedings of the Second International Conference on Genetic Algorithms*, 50-58. Cambridge, MA: Lawrence Erlbaum.

Smith, S. (1983) Flexible learning of problem solving heuristics through adaptive search. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 422-425. Karlsruche, Germany: William Kaufmann.

Spears, W. M., and De Jong, K. A (1991) On the virtues of uniform crossover. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 230-236. La Jolla, CA: Morgan Kaufmann.

Spears, W. M. (1992a) Crossover or mutation? *Proceedings of the Foundations of Genetic Algorithms Workshop*, Vail, Colorado: Morgan Kaufmann.

Spears, W. M. (1992b) Adapting crossover in a genetic algorithm. *Naval Research Laboratory AI Center Report AIC-92-025*. Washington, DC 20375 USA.

Vose, M. D., & Liepins, G. E. (1991) Schema disruption. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 237-242. La Jolla, CA: Morgan Kaufmann.

Whitley, D. (1989) The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. *Proceedings of the Third International Conference on Genetic Algorithms*, 116-121. Fairfax, VA: Morgan Kaufmann.

Whitley, D., Starkweather, T., & Fuquay, D. (1989) Scheduling problems and traveling salesmen: the genetic edge recombination operator. *Proceedings of the Third International Conference on Genetic Algorithms*, 133-140. Fairfax, VA: Morgan Kaufmann.

Whitley, D., Mathias, K., & Fitzhorn, P. (1991) Delta coding: an iterative search strategy for genetic algorithms. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 77-84. La Jolla, CA: Morgan Kaufmann.

Whitley, D. (ed.) (1992) *Proceedings of the Foundations of Genetic Algorithms Workshop*. Vail, CO: Morgan Kaufmann.