An Introduction to Evolutionary Computation

Talib S. Hussain

Department of Computing and Information Science
Queen’s University, Kingston, Ont. K7L 3N6

hussain@qucis.queensu.ca

Abstract

Researchers in many fields are faced with computational problems in which a great number of solutions are possible and finding an optimal or even a sufficiently good one is difficult. A variety of search techniques have been developed for exploring such problem spaces, and a promising approach has been the use of algorithms based upon the principles of natural evolution. This tutorial will introduce the basic principles underlying most evolutionary algorithms, as well as some of the key details of the four most popular methods: genetic algorithms, genetic programming, evolutionary strategies, and evolutionary programming. The aim of the tutorial is to introduce the participants to the jargon and principles of the field of evolutionary computation, and to encourage the participants to consider the potential of applying evolutionary optimization techniques in their own research.

1. Introduction

An important area in current research is the development and application of search techniques based upon the principles of natural evolution. Most readers, through the popular literature and typical Western educational experience, are probably aware of the basic concepts of evolution. In particular, the principle of the '*survival of the fittest*' proposed by Charles Darwin (1859) has especially captured the popular imagination. We shall use this as a starting point in introducing evolutionary computation.

The theory of natural selection proposes that the plants and animals that exist today are the result of millions of years of adaptation to the demands of the environment. At any given time, a number of different organisms may co-exist and compete for the same resources in an ecosystem. The organisms that are most capable of acquiring resources and successfully procreating are the ones whose descendants will tend to be numerous in the future. Organisms that are less capable, for whatever reason, will tend to have few or no descendants in the future. The former are said to be more *fit* than the latter, and the distinguishing characteristics that caused the former to be more fit are said to be *selected for* over the characteristics of the latter. Over time, the entire population of the ecosystem is said to *evolve* to contain organisms that, on average, are more fit than those of previous generations of the population because they exhibit more of those characteristics that tend to promote survival.

Evolutionary computation techniques abstract these evolutionary principles into algorithms that may be used to search for optimal solutions to a problem. In a search algorithm, a number of possible solutions to a problem are available and the task is to find the best solution possible in a fixed amount of time. For a search space with only a small number of possible solutions, all the solutions can be examined in a reasonable amount of time and the optimal one found. This *exhaustive search*, however, quickly becomes impractical as the search space grows in size. Traditional search algorithms randomly sample (e.g., *random walk*) or heuristically sample (e.g., *gradient descent*) the search space one solution at a time in the hopes of finding the optimal solution. The key aspect distinguishing an evolutionary search algorithm from such traditional algorithms is that it is *population-based*. Through the adaptation of successive generations of a large number of individuals, an evolutionary algorithm performs an efficient directed search. Evolutionary search is generally better than random search and is not susceptible to the hill-climbing behaviors of gradient-based search.

2. Basic Evolutionary Computation

In an evolutionary algorithm, a *representation scheme* is chosen by the researcher to define the set of solutions that form the search space for the algorithm. A number of individual solutions are created to form an *initial population*. The following steps are then repeated iteratively until a solution has been found which satisfies a pre-defined *termination criterion*. Each individual is evaluated using a *fitness function* that is specific to the problem being solved. Based upon their fitness values, a number of individuals are chosen to be *parents*. New individuals, or *offspring*, are produced from those parents using *reproduction operators*. The fitness values of those offspring are determined. Finally, survivors are selected from the old population and the offspring to form the new population of the next *generation*.

The mechanisms determining which and how many parents to select, how many offspring to create, and which individuals will survive into the next generation together represent a *selection method*. Many different selection methods have been proposed in the literature, and they vary in complexity. Typically, though, most selection methods ensure that the population of each generation is the same size.

The remainder of the paper presents the traditional definitions of the four most common evolutionary algorithms: genetic algorithms (Holland, 1975), genetic programming (Koza, 1992, 1994), evolutionary strategies (Rechenberg, 1973), and evolutionary programming (Fogel et al., 1966). The traditional differences between the approaches involve the nature of the representation schemes, the reproduction operators, and the selection methods.

3. Genetic Algorithms

The most popular technique in evolutionary computation research has been the *genetic algorithm*. In the traditional genetic algorithm, the representation used is a *fixed-length bit string*. Each position in the string is assumed to represent a particular feature of an individual, and the value stored in that position represents how that feature is expressed in the solution. Usually, the string is "evaluated as a collection of *structural* features of a solution that have little or no interactions" (Angeline, 1996, p. 4). The analogy may be drawn directly to genes in biological organisms. Each gene represents an entity that is structurally independent of other genes.

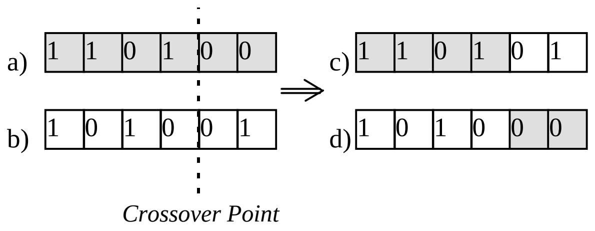

Figure 1: Bit-String Crossover of Parents a & b to form Offspring c & d

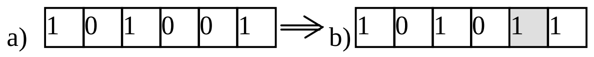

Figure 2: Bit-Flipping Mutation of Parent a to form Offspring b

The main reproduction operator used is *bit-string crossover*, in which two strings are used as parents and new individuals are formed by swapping a sub-sequence between the two strings (see Figure 1). Another popular operator is *bit-flipping mutation*, in which a single bit in the string is flipped to form a new offspring string (see Figure 2). A variety of other operators have also been developed, but are used less frequently (e.g., *inversion*, in which a subsequence in the bit string is reversed). A primary distinction that may be made between the various operators is whether or not they introduce any new information into the population. Crossover, for example, does not while mutation does. All operators are also constrained to manipulate the string in a manner consistent with the structural interpretation of genes. For example, two genes at the same location on two strings may be swapped between parents, but not combined based on their values.

Traditionally, individuals are selected to be parents *probabilistically* based upon their fitness values, and the offspring that are created replace the parents. For example, if N parents are selected, then N offspring are generated which replace the parents in the next generation.

4. Genetic Programming

An increasingly popular technique is that of *genetic programming*. In a standard genetic program, the representation used is a variable-sized tree of functions and values. Each leaf in the tree is a label from an available set of value labels. Each internal node in the tree is label from an available set of function labels. The entire tree corresponds to a single function that may be evaluated. Typically, the tree is evaluated in a left-most depth-first manner. A leaf is evaluated as the corresponding value. A function is evaluated using as arguments the result of the evaluation of its children.

Genetic algorithms and genetic programming are similar in most other respects, except that the reproduction operators are tailored to a tree representation. The most commonly used operator is *subtree crossover*, in which an entire subtree is swapped between two parents (see Figure 3). In a standard genetic program, all values and functions are assumed to return the same type, although functions may vary in the number of arguments they take. This *closure* principle (Koza, 1994) allows any subtree to be considered structurally on par with any other subtree, and ensures that operators such as sub-tree crossover will always produce legal offspring.

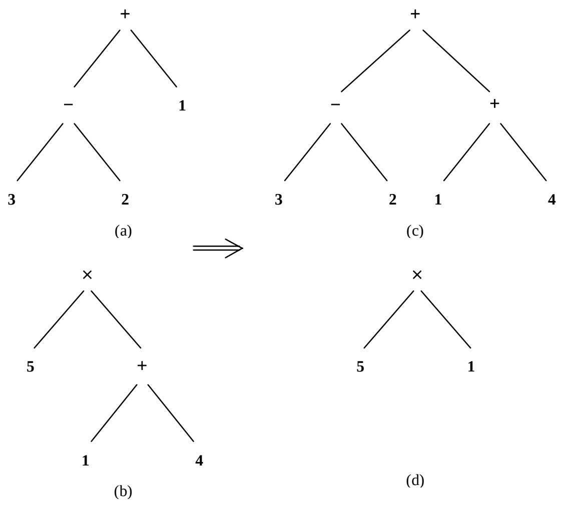

Figure 3: Subtree Crossover of Parents a & b to form Offspring c & d

5. Evolutionary Strategies

In evolutionary strategies, the representation used is a fixed-length real-valued vector. As with the bit-strings of genetic algorithms, each position in the vector corresponds to a feature of the individual. However, the features are considered to be behavioral rather than structural. "Consequently, arbitrary non-linear interactions between features during evaluation are expected which forces a more holistic approach to evolving solutions” (Angeline, 1996, p. 4).

The main reproduction operator in evolutionary strategies is *Gaussian mutation*, in which a random value from a Gaussian distribution is added to each element of an individual’s vector to create a new offspring (see Figure 4). Another operator that is used is *intermediate recombination*, in which the vectors of two parents are averaged together, element by element, to form a new offspring (see Figure 5). The effects of these operators reflect the behavioral as opposed to structural interpretation of the representation since knowledge of the values of vector elements is used to derive new vector elements.

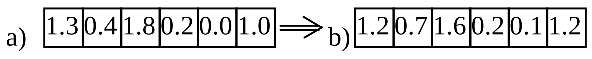

Figure 4: Gaussian Mutation of Parent a to form Offspring b

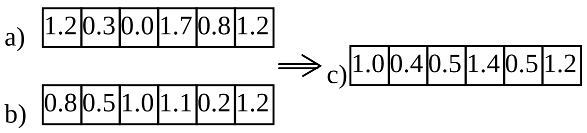

Figure 5: Intermediate Recombination of Parents a & b to form Offspring c

The selection of parents to form offspring is less constrained than it is in genetic algorithms and genetic programming. For instance, due to the nature of the representation, it is easy to average vectors from many individuals to form a single offspring. In a typical evolutionary strategy, N parents are selected uniformly randomly (i.e., not based upon fitness), more than N offspring are generated through the use of recombination, and then N survivors are selected deterministically. The survivors are chosen either from the best N offspring (i.e., no parents survive) or from the best N parents and offspring (Spears et al., 1993).

6. Evolutionary Programming

The representations used in evolutionary programming are typically tailored to the problem domain (Spears et al., 1993). One representation commonly used is a fixed-length real-valued vector.

The primary difference between evolutionary programming and the previous approaches is that no exchange of material between individuals in the population is made. Thus, only mutation operators are used. For real-valued vector representations, evolutionary programming is very similar to evolutionary strategies without recombination.

A typical selection method is to select all the individuals in the population to be the N parents, to mutate each parent to form N offspring, and to probabilistically select, based upon fitness, N survivors from the total 2N individuals to form the next generation.

7. Current Issues

In current research, the line distinguishing these different approaches has started to blur. Researchers in each technique have begun to examine more complex representation schemes and to apply a variety of selection methods. Many genetic algorithm researchers are examining the use variable-length representations and analyzing how such representations grow in size over the course of evolution (Wu & Lindsay, 1996). Many genetic algorithms now use selection methods, such as *elitist recombination*, in which parents compete with their offspring for survival into the next generation (Thierens, 1997). Some genetic programming researchers have begun to examine the effects of allowing multiple types of functions and values into the representation. The benefits of such strongly typed genetic programming are only beginning to be explored (Haynes et al., 1996).

References

Angeline, P.J. (1996) “Genetic programming’s continued evolution,” Chapter 1 in K.E. Kinnear, Jr. and P.J. Angeline (Eds.), *Advances in Genetic Programming 2*. Cambridge, MA: MIT Press, p. 1 -20.

Darwin, C. (1859) *On the Origin of Species by Means of Natural Selection*. London: John Murray.

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

Haynes, T.D., Schoenefeld, D.A. & Wainwright, R.L. (1996) “Type inheritance in strongly typed genetic programming,” Chapter 18 in K.E. Kinnear, Jr. and P.J. Angeline (Eds.), *Advances in Genetic Programming 2*. Cambridge, MA: MIT Press. p. 359-376.

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

Koza, J.R. (1992) *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press.

Koza, J.R. (1994) *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, MA: MIT Press.

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

Spears, W.M., DeJong, K.A., Back, T., Fogel, D.B., & deGaris, H. (1993) “An overview of evolutionary computation,” *Proceedings of the 1993 European Conference on Machine Learning*.

Thierens, D. (1997) “Selection schemes, elitist recombination, and selection intensity,” *Proceedings of the Seventh International Conference on Genetic Algorithms*. San Francisco, CA: Morgan Kauffman, p. 152-159.

Wu, A. & Lindsay, R.K. (1996) “A survey of intron research in genetics,” In Voigt, H., Ebelin, W., Rechenberg, I., & Schwefel, H. (Eds.), *Parallel Problem Solving from Nature - PPSN IV*. Berlin: Springer-Verlag, p. 101-110.