Genetic Algorithms and Evolutionary Computing

Darrell Whitley
Computer Science Department, Colorado State University
Fort Collins, CO 80523 whitley@cs.colostate.edu

1 Introduction

Genetic Algorithms are a family of computational models inspired by evolution. Other genetic and evolutionary algorithms include *Evolution Strategies*, *Genetic Programming* and *Evolutionary Programming*.

Genetic algorithms have been used for parameter optimization problems, scheduling applications and design optimization. In terms of fielded applications, genetic algorithms have been used to optimize a paper production process, to schedule the assembly lines of major automobile, truck and tractor manufacturing companies, and genetic algorithms have been used to design turbine engines currently used on commercial passenger aircraft.

Genetic and evolutionary algorithms encode a potential solution to a specific problem on a chromosome-like data structure and apply recombination operators to these structures so as to preserve critical information. Mutation operators are also used to alter potential solutions.

An implementation of a genetic algorithm begins with a population of (typically random) artificial chromosomes. One then evaluates these structures and allocates reproductive opportunities in such a way that those chromosomes that represent a better solution to the target problem are given more chances to "reproduce" than those chromosomes representing poorer solutions. The "goodness" of a solution may be computed by comparing its evaluation against the population average, or it may be a function of the rank of that individual in the population relative to other solutions.

The term *genetic algorithm* can have two meanings. In a strict interpretation, *the genetic algorithm* refers to a model introduced and investigated by John Holland [7, 8] and by his Ph.D. students [6] [3]. Most of the existing theory for genetic algorithms applies either solely or primarily to the model introduced by Holland. In a broader sense, a genetic algorithm is any population-based model that uses selection and recombination operators and mutation operators to generate new sample points in a search space.

Evolutionary algorithms do not use gradient information. Thus, solutions may be evaluated using a mathematical function, or by using a computer simula-

tion. For example, when genetic algorithms were used to design aircraft engines, a simulation used to determine the fuel savings associated with different designs. This makes is possible to use genetic and evolutionary algorithms for applications where other mathematical optimization techniques are not appropriate. These algorithms are particularly useful for ill-structured search problems that are characterized by having a large number of local optima.

1.1 Encodings, Operators and Evaluation

The problem encoding and the evaluation function both interact to determine the best recombination and mutation operators to use.

For example, we may wish to minimize (or maximize) some function

$$F(X_1, X_2, ..., X_M)$$

where X_i is an input parameter. For this kind of parameter problem, the encoding may be composed of bit strings or real values.

On the other hand, we might want to schedule a manufacturing line. For this kind of combinatorial optimization problem, a permutation based representation might be used, where the permutation represents the order in which tasks are scheduled. Also, consider a routing problem, where we must decide the order in which a truck make deliveries to some large number of customers.

Once an encoding is choosen, appropriate recombination and mutation operators must be developed.

For parameter optimization a simple binary string can be used:

1101001100101101.

The string could broken apart and decoded into parameters such as values for temperature or pressures or material-ratios in the paper production process for example. New solutions are generated by recombining two parent strings. Consider the string 1101001100101101 and another binary string, yxyyxyxxyyyyxyxxy, where the values 0 and 1 are denoted by x and y. Using two randomly chosen crossover points, recombination might occur as follows.

```
11010 \/ 01100 \/ 101101
yxyyx /\ yxxyy /\ yxyxxy
```

Swapping the fragments between the two parents produces the following offspring.

```
11010yxxyy10110      and       yxyyx01100yxyxxy
```

After recombination, mutation can be applied. Typically the mutation rate is low so that only one bit changes in the offspring.

Integer and real values can be represented as bit strings in various ways. A discretization of real-values can be mapped to integers, and integers can be

mapped to bits using standard Binary Coded Decimal representations. However, standard binary encodings have certain disadvantages. For example, the integers 15 and 16 are encoded as 01111 and 10000. Note that these values are neighbors in integer space but are not neighbors in terms of the bit-space or *Hamming* neighborhood associated with the standard Binary representation. In fact, these strings form a Hamming cliff: adjacent integers are represented by complementary bit strings and thus share no bits in common.

It may be desirable to use a different bit encoding where adjacent integers are represented by bit strings that are neighbors in Hamming space and thus differ by a single bit. A *Gray code*, by definition, is any bit representation where adjacent integers are represented by bit strings that differ by a single bit. There are in fact many Gray codes. The actual number of Gray codes is unknown. Standard Binary Reflected Gray code is most commonly used. While the use of Binary Code Decimal representations can induce new local optima in Hamming space that do not exists in the original function, Gray codes are always guaranteed to produce a representation in Hamming space where the number of local optima is less than or equal to the number of optima in the original real-valued or integer function representation [13].

Some researchers argue that if the "natural" representation of a problem is integer, or real valued, then that representation should be used. Recombination and mutation operators can certainly be applied to real valued strings. And some forms of evolutionary algorithms, such as Evolution Strategies, are designed specifically to work with real valued representations of parameter optimization problems.

Scheduling problems may use complex representations, but often can be represented using premutations indicating the sequence in which tasks or events occur. Simple cut and swap recombination operations work well for parameter optimization problems, but more complex operators are needed for permutation representations. Furthermore, both scheduling problems and routing problems may use permutation representations, but the recombination and mutation operators may be very different. For routing problems, the *adjacency* of events is often important, while in scheduling the *relative order of events* is important. *Adjacency* implies the evaluation function is sensitive to the fact that event A occurs immediately before (or after) event B in a permutation such as $KQABCD$. *Relative Order* implies the evaluation function is sensitive to the fact that event A occurs at some time before (or after) event B, but it need not be immediately before (or after), as in the permutation such as $QAKCBD$. Many operators have been developed to try to maintain either adjaceny or relative order, and operators that work well on one type of permutation-based problem (e.g., scheduling) often perform poorly on a different type of permutation-based problem (e.g., routing or the Traveling Salesman Problem) [2].

1.2 Beyond Simple Genetic Algorithms

While genetic algorithms were being developed in the United States between 1970 and 1990, German researchers Ingo Rechenberg [10] and Hans-Paul Schwe-

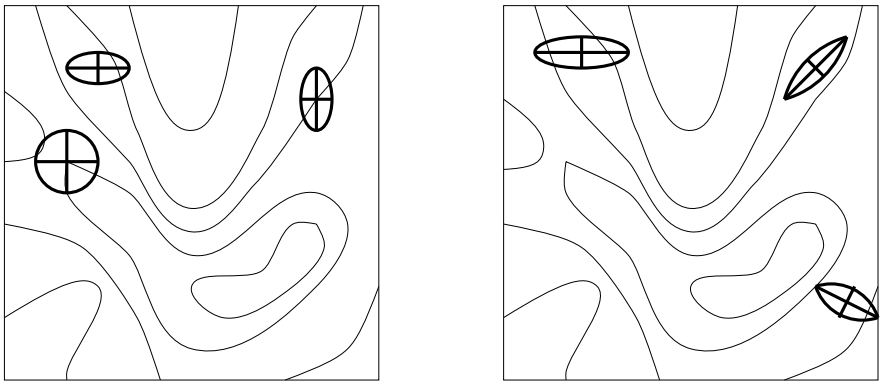

Figure 1: A two dimensional search space $F(X_1, X_2)$ is shown, along with elipses representing chromosomes in the population. Contours represent changes in evaluation. In the leftmost graph, simple mutation used with only a step-size in each direction. In the rightmost graph, adaptive rotation parameters are also used.

fel [11, 12] were developing *Evolution Strategies*.

Evolution Strategies are also a population-based form of search that have largely been developed for parameter optimization problems. Evolution Strategies generally use real-valued parameter encodings and emphasize the use of mutation rather than recombination. Evolution Strategies use "strategy parameters" that control the mutation step size for each parameter on the chromosome. These strategy parameters are also encoded onto the chromosome and evolve along side the regular parameters. The strategy parameters are not really fixed step sizes, but rather standard deviations. Mutation is usually defined in terms of a sampling distribution around the current members of the population; a Gausian or log-normal distribution might be used for each parameter and the strategy parameters would be the standand deviations associated with the distributions. Thus, the actual step is choosen stochastically by selecting the step-size based on the corresponding probability distribution. The mutation step-size is different in each dimension of the search space. In addition, a correlation or rotation parameter can also be used for each pair of parameters. This rotation allows the search to not only adapt the step size used by mutation, but also allows it to adapt the direction of mutation [1, 12]. This idea is illustrated in Figure 1.

Encoding the strategy parameters onto the chromosome also makes Evolution Strategies self adaptive. Addition strategy parameters are also sometimes added onto the chromosome that control the direction of new mutations. Recombination operators, such as swapping parameter values or averaging parameter values, are also sometimes used but recombination does not have the same

primary role that it does in genetic algorithms.

Evolution Strategies (ES) have a rich notation for describing different types of evolutionary algorithms. For example, parents may "die" upon reproducing and are replaced by their offspring. This happens in Holland's standard genetic algorithms. This is also known as a $(\mu, \lambda) - ES$. The population with μ number of parents is replaced by λ offspring. On the other hand, some forms of the algorithm allow the best members of both the parent population and the offsprings to survive into the next generation. This is known as a $(\mu + \lambda) - ES$. This strategy is also used in what has come to be known as "steady-state" genetic algorithms [14].

Evolutionary Programming as practiced today is a reincarnation of earlier evolutionary computing methods developed by Lawrence Fogel, A.J. Owens and M.J. Walsh in the 1960's [5]. During the 1960's evolutionary programming used mutation to change finite state machines. The main idea behind evolutionary programming is to search in "phenotype" (or behavior) space rather than searching in "genotype" space (the space of genes that indirectly control for behavior after decoding and development). Hence, operators act directly on the finite state machines as opposed to some special genotype or encoded representation. In the 1990's David Fogel, reintroduced evolutionary programming as a general parameter optimziation method [4], but for all practical purposes, the modernized evolutionary programming algorithms are virtually identical to evolution strategies, except that recombination is never used. Real valued encoding and mutation controlled by strategy parameters are used instead.

The most recent addition to the family of Genetic and Evolutionary Algorithms is *Genetic Programming* [9]. Genetic Programming is used to evolve computer programs. Thus, the chromosomes are programs. Traditionally these programs are in the language "Lisp." Lisp programs take the form of "s-expressions." An s-expression can be defined recursively as

```
s-expression =  (operator  s-expression  s-expression ... s-expression)
s-expression =  primitive
```

Because each recursively defined s-expression is itself an executable form, one can recombine subparts of an s-expression and still produce a syntactically legal and executable form.

For example

```
s-expression1 =  (add  1 (add 7 8) 5)
s-expression2 =  (multiply  2  (multiply 2 X)
```

might be recombined to produce

```
s-expression3 =  (add  (add 7 8) (multiply 2 X))
```

One can evolve polynomial forms, for example, that can be used for general function approximation. The evaluation function, in this case, might take the form of a least-mean-square error function that matches the output of the evolving polynomials against known sample training data.

Finally, genetic and evolutionary algorithms lend themselves to highly parallel implementation. Members of the population can be selected, recombined, mutated and evaluated in parallel. Sometimes subpopulations are evolved in parallel producing an *Island Model Evolutionary Algorithm.* Surprisingly, evolving distinct subpopulations that only occasionally exchange chromosomes can often result in faster and more robust optimization than using a single large population. The ability to execute these algorithms using a high degree of parallelism can make them well suited for optimization problems where good results are needed quickly [14].

References

[1] T. Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, 1996.

[2] Lawrence Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[3] Ken DeJong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Department of Computer and Communication Sciences, Ann Arbor, Michigan, 1975.

[4] D. B. Fogel. Evolutionary programming: an introduction and some current directions. *Statistics and Computing*, 4:113–130, 1994.

[5] L.J. Fogel, Owens A.J., and M.J. Walsh. *Artificial Intelligence Through Simulated Evolution.* John Wiley, 1966.

[6] David Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[7] John Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[8] John H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, MA, second edition, 1992.

[9] John Koza. *Genetic programming: A paradigm for genetically breeding computer population of computer programs to solve problems.* MIT Press, Cambridge, MA, 1992.

[10] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboof, Stuttgart, 1973.

[11] Hans-Paul Schwefel. *Numerical Optimization of Computer Models.* Wiley, 1981.

[12] Hans-Paul Schwefel. *Evolution and Optimum Seeking.* Wiley, 1995.

[13] Darrell Whitley, Keith Mathias, Soraya Rana, and John Dzubera. Building Better Test Functions. In L. Eshelman, editor, *Proc. of the 6th Int'l. Conf. on GAs*. Morgan Kaufmann, 1995.

[14] L. Darrell Whitley. A Genetic Algorithm Tutorial. *Statistics and Computing*, 4:65–85, 1994.