A Genetic Algorithm Programming Environment: Splicer

Steven E. Bayer
Space Systems Division
The MITRE Corporation
1120 NASA Road One, Suite 600
Houston, TX 77058
bayer@mpad.span.nasa.gov

Lui Wang
Software Technology Branch
Lyndon B. Johnson Space Center
NASA
Houston, TX 77058
wang@mpad.span.nasa.gov

Abstract

Genetic algorithms are highly parallel, mathematical, adaptive search procedures loosely based on the processes of natural genetics and Darwinian survival of the fittest. Genetic algorithms have been used to solve parameter optimization problems and for machine learning. This paper introduces basic genetic algorithm concepts, discusses genetic algorithm applications, and presents results of a project to develop a software tool—a genetic algorithm programming environment—called Splicer.

1 Introduction

Genetic algorithms (GAs) were pioneered by John Holland in his research on adaptation in natural and artificial systems (1). This research outlined a logical theory of adaptive systems. In essence, biological adaptive systems strive to optimize single individuals or entire species for specific environments to increase the chance of survival. Holland simulated the methods used when biological systems adapt to their environment in computer software models—the genetic algorithms—to solve optimization and machine learning problems. The following paragraphs briefly discuss two types of adaptation strategies that are observed in many biological systems and that inspired the basic framework of genetic algorithms.

1.1 Adaptation

One form of adaptation pertains to the way an individual changes within its environment to promote survival. Examples include the development of antibodies specific to certain diseases and the enlargement of muscles needed for daily activities. The way we learn, and the neural changes that accompany learning, is another example of how an individual adapts within its environment. The effects of this form of adaptation are not imprinted on the genome (the genetic makeup of a species); that is, they are not passed on from generation to generation. On the other hand, individual adaptation does promote the survival of the individual within an environment—survival of the fittest—and enhances that individual's net reproductive advantage through a natural selection where fitter members of a population are more likely to reproduce.

All species have used adaptive search for millions of years, through an evolutionary search process, to improve the way a species lives and survives within its environment. Therefore, adaptation also refers to evolution and modification of an entire species to fit its environment. This is the process of making a species environmentally fit. An appropriate example can be seen in the way many plant species have evolved their flower to resemble a female bee or wasp that attracts the male counterpart and promotes pollination. This evolutionary or species adaptation is imprinted on the genome and is passed on to subsequent generations.

Thus natural, biological systems continuously use adaptive search to improve genomes—that is, to improve the species—and to promote the survival of fitter individuals and genomes through natural selection.

1.2 Genetic Algorithms

Genetic algorithms are highly parallel, mathematical, adaptive search procedures (i.e., problem-solving methods) based loosely on the processes of natural genetics and Darwinian survival of the fittest. These algorithms apply genetically-inspired operators to populations of potential solutions in an iterative fashion, creating new populations while searching for an optimal (or near-optimal) solution to the problem at hand. Population is a key word here: the fact that many points in the space are searched in parallel sets genetic algorithms apart from other search operators. Another important characteristic of genetic algo-
gorithms is the fact that they are very effective when search-
ing (e.g., optimizing) function spaces that are not smooth
or continuous—functions which are very difficult (or im-
possible) to search using calculus based methods. Genetic
algorithms are also blind: that is, they know nothing of
the problem being solved other than payoff or fitness (i.e.,
objective function) information.

The basic iterative model of the genetic algorithms is
shown in figure 1. A new population is created from an
existing population by means of evaluation, selection, and
reproduction. This process repeats itself until the popula-
tion converges on an optimal solution or some other
stopping condition is reached.

![Figure 1. The Iterative Genetic Algorithm Model.]

The initial population consists a set of individuals
(i.e., potential solutions) generated randomly or heuristi-
cally. In the classical genetic algorithm, each member is
represented by a fixed-length binary string of bits (a chro-
mosome) that encodes the parameters of the problem.
This encoded string can be decoded to give the integer val-
ues for these parameters. (Other coding schemes—e.g.,
permutations for ordering problems—are also used.) The
two toughest challenges that must be faced when using
genetic algorithms are 1) representing the problem using a
coding that is appropriate, and 2) finding a set of genetic
operators (mainly, crossover and mutation) that are
appropriate for the chosen coding scheme.

Once the initial population has been created, the eval-
uation phase begins. The genetic algorithms require that
members of the population can be differentiated according
to goodness or fitness. The members that are more fit are
given a higher probability of participating during the se-
lection and reproduction phases. Fitness is measured by
decoding a chromosome and using the decoded parameters
as input to an objective function (the objective function is
specific to the problem being solved). The value returned
by the objective function (or some transformation of it) is
used as the fitness value.

During the selection phase, the population members
are given a target sampling rate that is based on fitness
and determines how many times a member will mate dur-
ing this generation—that is, how many offspring from this
individual will be created in the next population. The tar-
get sampling rate (usually not a whole number) must be
transformed into an integer number of matings, a mating
count, for each individual. There are many ways of deter-
mining the target sampling rate and the actual number of
matings. Suffice it to say that individuals that are more
fit are given a reproductive advantage over less fit mem-
bers.

During the reproduction phase, two members are cho-
sen randomly from the mating pool (i.e., the group of
population members with non-zero mating counts) and
genetic operators (crossover and mutation) are applied to
them (during the recombination phase) to produce two
new members for the next population. This process is re-
peated until the next population is filled, constituting on
generation. The recombination phase usually involves
two operators: crossover and mutation.

A simple crossover operation is illustrated in figure
2. During crossover, the two parents exchange substring
information (genetic material) at a random position in the
chromosomes to produce two new strings. The crossover
operation searches for better building blocks within the
genetic material which combine to create optimal or near-
optimal problem parameters and, therefore, problem
solutions, when the string is decoded. The goal here is to
create better individuals, and a better population over time,
by combining material from pairs of fitter members from
the current population. Crossover occurs according to a
crossover probability, usually between 0.5 and 1.0.

![Figure 2. The Crossover Operation.]

The mutation operation, shown in figure 3, is a sec-
ondary genetic algorithm. It is used to maintain diversity
in the population—that is, to keep the population from
prematurely converging on one solution—and to create ge-
netic material that may not be present in the current popu-
ation. The mechanics of the mutation operation are sim-
ple: for each position (i.e., each bit) in a string created
during crossover, change the value at that position accord-
ing to a mutation probability. The mutation probability
is usually very low—less than 0.05.
1.3 Genetic Algorithm Applications

Since genetic algorithms provide a set of efficient, domain-independent search methods, they have been used for a wide range of applications: from science and engineering to business and social sciences applications. The following sections briefly describe several of these applications.

**Engineering:** Goldberg (3) applied genetic algorithms and classifier systems to optimization and machine learning problems in natural gas pipeline control. He focused on a 10-compressor, 10-pipe, steady-state, serial pipeline problem. The object was to minimize the power consumed subject to maximum and minimum pressure and pressure ratio constraints. Goldberg and Samtani (4) used a simple genetic algorithm to optimize a 10-member plane truss. The objective was to minimize the weight of the structure subject to maximum and minimum stress constraints on each member. In both cases, optimal or near-optimal results were obtained.

**Medical Image Processing:** Fitzpatrick, et al. (5), used genetic algorithms to perform image registration for an arterial examination system known as digital subtraction angiography. Using this system, a physician examines arteries using two x-rays: one taken of the artery unaltered and one taken after injection of a dye into the artery. The two x-ray images are subtracted one pixel at a time; the result is a picture of the interior wall of the artery. Movement of the artery between the time each x-ray is taken results in a distorted image. Fitzpatrick, et al., used genetic algorithms to find a set of equations that transform or register the two images.

**Robot Path Planning:** A Mobile Transporter system is being designed for on-orbit use with Space Station Freedom which will be capable of traversing the station's truss structure. The Mobile Transporter's function is to facilitate space station maintenance tasks and transportation of material around the station. The Software Technology Branch (STB) has investigated the use of genetic algorithms for Mobile Transporter path planning (6). The objectives of these activities are to produce an optimum trajectory for the Mobile Transporter that avoids collisions with objects attached to the truss and to minimize the length of the path between the Mobile Transporter and the target position.

**Machine Learning:** Genetic algorithms have been used in an area of machine learning called classifier systems. Classifier systems learn if-then production rules that guide the performance of a production system. Holland has used classifier systems in studies of economic models, specifically mathematical stock market models. The genetic algorithm creates new rules for trading stocks.

2 The Splicer Project

This section introduces the Splicer Project. It presents background material and discusses the objectives of the project, the approach taken, results to date, and current status.

2.1 Background

The Splicer Project is a project within the Software Technology Branch at NASA's Johnson Space Center. The purpose of the project is to develop a tool that will enable the widespread use of genetic algorithm technology. The charter of the Software Technology Branch is to develop and/or acquire software tools for emerging technologies. Genetic algorithms are just one of the many technologies being investigated within the Software Technology Branch: other areas and tools are expert systems (CLIPS), neural networks (NETS), fuzzy logic, scheduling (COMPASS), software reuse, and intelligent computer-aided training.

The MITRE Corporation supports the Software Technology Branch on multiple projects and is responsible for evaluating the viability and robustness of genetic algorithms and for supporting the Software Technology Branch with respect to the development and acquisition of software tools related to this technology.

2.2 Objectives

The Software Technology Branch is interested in applying genetic algorithms within various domains: e.g., robot path planning and job shop scheduling. The original goal of the Splicer Project was to create a flexible, generic tool. As such, the tool would:

- Implement the basic genetic algorithms defined in the literature
- Define the interfaces for and allow users to develop interchangeable fitness modules
- Provide a graphic, event-driven user interface
Subsequent goals include the following:
- Distribution of the tool through COSMIC, the distributor of government-developed software
- Support for multiple computing platforms
- Extension of the tool for additional genetic algorithm functionality
- Use of the tool for genetic algorithm research
- Augmentation of the tool and its user interface for specific application domains

2.3 Approach

**Design:** The design chosen for the Splicer tool is shown in figure 4. This design consists of four components: a genetic algorithm kernel and three types of interchangeable libraries or modules: representation libraries, fitness modules, and user interface libraries.

A *genetic algorithm kernel* was developed that is independent of representation (i.e., problem encoding), fitness function, or user interface type. The GA kernel comprises all functions necessary for the manipulation of populations. These functions include the creation of populations and population members, the iterative population model, fitness scaling, parent selection and sampling, and the generation of population statistics. In addition, miscellaneous functions are included in the kernel (e.g., random number generators).

**Implementation:** The C programming language was chosen for portability and modularization. The original prototype was developed on an Apple Macintosh using Symantec's Think C. This included the development of the Macintosh user interface. The GA kernel, representation libraries, and fitness functions were then ported to a Sun 3/80 and SPARC. An X Window System user interface was then developed using X and the Hewlett-Packard Widget Set.

2.4 Results

This section presents the genetic operators and other features provided by the Splicer genetic algorithm tool. This is done using brief descriptions and, in some cases, screen shots from the Macintosh interface (components of the X Window System interface are very similar). Once a fitness function has been developed, compiled, and linked with the Splicer libraries, the resulting application can be executed to solve the particular parameter optimization problem. Subsequent steps would include defining control parameters, running the genetic operators, displaying information during the run, changing parameters as the genetic algorithms execute, and saving results to disk.

**Control Parameters:** The Control Parameters dialog box, shown in figure 5, allows the user to set the values of multiple parameters that control the operation of the Splicer tool (e.g., population size, crossover operator, mutation probability). This is accomplished in two ways: numeric parameters have buttons associated with them that pop up dialog boxes to allow the user to enter a new value; genetic operators (e.g., the fitness scaling operator).
have pop up menus associated with them that allow the user to select from a list of operators.

<table>
<thead>
<tr>
<th>Control Parameters</th>
<th>Parameter Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Parameters:</td>
<td>5</td>
</tr>
<tr>
<td>Population Size:</td>
<td>50</td>
</tr>
<tr>
<td>Scaling Operator:</td>
<td>None</td>
</tr>
<tr>
<td>Sharing Operator:</td>
<td>None</td>
</tr>
<tr>
<td>Selection Operator:</td>
<td>Proportional</td>
</tr>
<tr>
<td>Sampling Operator:</td>
<td>Ranking Method</td>
</tr>
<tr>
<td>Crossover Operator:</td>
<td>Single Point</td>
</tr>
<tr>
<td>Mutation Operator:</td>
<td>Point</td>
</tr>
<tr>
<td>Crossover Probability:</td>
<td>0.6</td>
</tr>
<tr>
<td>Mutation Probability:</td>
<td>0.0333</td>
</tr>
<tr>
<td>Random Number Seed:</td>
<td>-1</td>
</tr>
</tbody>
</table>

Figure 5. Control Parameters Dialog Box.

The Parameter Characteristics button on the Control Parameters dialog box is used to pop up another dialog box that allows the characteristics of the individual strings (i.e., members of the population, sometimes called "chromosomes") to be changed—for permutations there are no characteristics to change, therefore this button is disabled. The Parameter Characteristics dialog box for binary strings is shown in figure 6. This dialog box allows the user to specify the number of problem parameters and their size in number of bits. It also allows the parameter values to be normalized during decoding to create floating point numbers.

<table>
<thead>
<tr>
<th>Parameter Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Parameters: 5</td>
</tr>
<tr>
<td>Normalize Parameter Values</td>
</tr>
<tr>
<td>Parameters Are All The Same Size</td>
</tr>
<tr>
<td>Parameter Sizes:</td>
</tr>
<tr>
<td>Enter</td>
</tr>
</tbody>
</table>

Figure 6. The Parameter Characteristics Dialog Box.

Program Control: The operation of the Splicer program is controlled using options on the Control menu (as well as other menus). The Control Parameters dialog box is displayed using the Set Control Parameters... option. To create the initial, random population, the Create Population option is used. The Run option starts the execution of the genetic algorithms on the existing population. To begin again with a clean slate, the Reinitialize... option is used.

Operators: The Splicer program provides multiple operators for fitness scaling, parent selection and sampling, crossover, and mutation. These operators can be changed at any time, even while the genetic algorithms are executing, and are selected using the Operators menu or the buttons on the Control Parameters dialog box.

Fitness Scaling: Splicer provides a linear fitness scaling option. Fitness scaling is useful near the end of a genetic algorithm run when all members of the population have relatively high fitness and, therefore, have nearly equal reproductive potential (i.e., little selective advantage). Scaling spreads the fitness values equally over some range, allowing the members to be differentiated, and gives fitter members greater reproductive advantage. Scaling can be turned off.

Parent Selection: Selection assigns each member of the population a value representing its ideal participation in mating and reproduction; that is, selection assigns real-valued, target sampling rates. Parents can be selected using either proportional selection (i.e., using relative fitness values) or using linear rank selection (where population members are simply ranked according to fitness).

Parent Sampling: Target sampling rates, assigned during parent selection, must be converted to actual sampling rates. These are integers that specify the number of times each member will mate. Some members will mate several times, some will not mate at all. The members that mate for the "mating pool". There are many methods available for sampling parents and the following is a list of the ones Splicer offers:
- Deterministic
- Ranking Method (Stochastic Tournament)
- Roulette Wheel (Stochastic with Replacement)
- Stochastic Remainder with Replacement
- Stochastic Remainder without Replacement
- Stochastic Universal (Baker Selection)
- Tournament

Crossover: The availability of crossover operators depends upon the representation library being used. The binary string library provides the following crossover op-
operators: Single Point (described in the introduction), Two Point, and Uniform. Two Point crossover operates by selecting two random points within the parent strings with subsequent swapping of material between these two points. For Uniform crossover, a randomly generated mask is created (according to some probability) and individual bits are taken from the parents (which parent depends upon the value of corresponding bits in the mask) to create the children.

The following crossover operators are provided by the permutation library: Partially-Matched (PMX), Order, and Cycle. These operators are single-point crossover operators that ensure the resulting children are viable permutations. Using either library, crossover can be turned off by using the menu option (i.e., None) or by setting the crossover probability to zero.

Mutation: Similarly, the availability of mutation operators depends upon the representation library being used. The binary string library provides a Point mutation operator. Point mutation changes the value at each bit position in a string according to a coin toss that is weighted by the mutation probability.

The permutation library provides a PerMutation operator. PerMutation also operates on each position (cell) in the string (i.e., permutation) according to the mutation probability, but instead of changing the value at that position, the cell is swapped with another randomly chosen cell, thereby guaranteeing a viable permutation. Mutation can be turned off by using the menu option or by setting the mutation probability to zero.

Output: Various windows present information to the user as the genetic algorithms execute. These windows are described briefly in the following sections.

Statistics Window: The Statistics Window, shown in figure 7, displays the current generation number along with the objective function values for the best solution ever found (in all generations) and the best, average, and worst members of the current population.

Fitness Window: The Fitness Window, shown in figure 8, displays the fitness distribution of the current population. Fitness values are normalized, using the best ever fitness value, to create fitness values between 0.0 and 1.0. This interval is divided into ten bins (0.1 wide) and the percentage of the population in each bin is presented as a histogram.
Figure 9. The Objective Window.

Figure 10. The User Window.

Figure 11. The Trace Window.

Version 1.0 of the Splicer genetic algorithm tool (both Macintosh and X Windows) is available free to the Government and its contractors through the STB Help Desk: (713) 280-2233. Others may obtain Splicer (for a nominal fee) through COSMIC, the distributor of government-distributed software: (404) 542-3265.

References