A Genetic Algorithm problem solver for Archaeology

Carlos Reynoso (billyr@microsoft.com.ar) – Universidad de Buenos Aires

Edward Jezierski (edjez@microsoft.com) - Microsoft

Abstract

Generic algorithms are based on the mechanics of natural selection and genetics. They proved to be an important methodology in the development of search and machine-learning methods, but its relevance to archaeology is still to be tested. This paper consists in a basic introduction to the subject, and the presentation of a "workbench" program (to be distributed for free) implementing a general problem solver. The workbench requires a definition of the problem and a specification of how to solve it. A problem is defined in terms of data types (which specify how to decode alleles to phenotypical values), variables (of the specified data types) which define the genotype of an individual, and an evaluation function to maximize or minimize (the tool accepts a script, a series of IF-THEN-ELSE rules, or a compiled object). A solution configuration consist of which operators to use with what activation probabilities, a selection policy, a fitness mapping policy, an initial population size and a cut condition. The result produced is a collection of individuals with phenotypical and genotypical values that optimize the fitness/evaluation function until the cut condition is fulfilled. Archaeologists are invited to submit their problem definitions in order to test them.

Keys words: **modeling – archaeology – genetic algorithm – genetic computing – simulation.**

Genetic Algorithms Refresher

Genetic Algorithms (GAs) were initially developed by John Holland at the University oh Michigan in the seventies (Holland 1975). In the eighties they were combined with neural networks, resulting in a paradigm sometimes called "neural Darwinism". In the nineties it was usual to implement GAs in computer programs; today we talk of Genetic Programming, instead of GA. Anyway, GAs are used in search & optimization problems. Given a problem space in which to search and/or optimize, GAs will mimic natural evolutionary patterns to some degree in order to find individuals that meet a desired goal. Some of the most distinctive characteristics of GAs are:

- Genetic Algorithms are a stochastic, yet structured search (they are not a variation of Montecarlo approaches). Under the operators considered, GAs can be proven to converge to the solution.
- GAs tune the search based on a population of search points rather than only one search point. The position of a search point depends on the whole previous search population, not just a previous search point.
- Genetic Algorithms perform the search with no knowledge of the actual structure of the solution space but rather perform the search in a space which is a representation of the solution space

The following are common terms used when discussing Genetic Algorithms:

Population: A set of individuals at a given time,

Individual: A particular solution to a problem (it may not be the best solution),

Phenotype: The values that make up a solution,

Genotype: The 'genetic material' of an individual that can be transformed into a phenotype.

A comparison between natural and GA terminology is in order (Fig 1):

Natural	Genetic algorithm
Chromosome	**String**
Gene	**Feature, character or detector**
Allele	**Feature value**
Locus	**String position**
Genotype	**Structure**
Phenotype	**Parameter set, alternative solution, point (in the solution space)**
Epistasis	**Non linearity**

(Fig. 1)

GAs typically implement three operators: reproduction, crossover and mutation.

The characteristics outlined above and their underlying mathematics make Genetic Algorithms especially suitable for problems that:

- Have nonlinear search spaces – while GAs do searches in a space, the search is in a space that defines a representation of the solution, not the solution itself. This effectively decouples the solution space from the search space. For example, a genetic algorithm trying to find the highest number between 0 and 7 (obviously 7) could try to solve the problem in terms of finding the arrangement of 3 bits that yields the highest numbers. The GA in this case is unaware of the fact each bit sequence represents a natural number. It just asks an evaluator – what is the 'fitness

score' for an individual whose genotype is (whose genes say) "*101*"? The phenotype ("5" in this case) has no incidence over how the GA acts on the genes.

- Are highly polymodal – many points in the problem space may yield good solutions. This results to some extent from the point above. Going back to the example, in terms of search complexity, 0 and 1 are at the same Hamming distance than 0 and 4 – thus a search will have little tendency to fall in 'peaks' or 'valleys' of the problem space. This is also enhanced by the fact that the search is performed in terms of populations, not single points.

GAs have many other attributes whose relative importance depends on the application at hand.

Following is a quick summary of a generic Genetic Algorithm process. This will help understand the design and implementation processes that had to be undergone to do problem solving with Genetic Algorithms.

Process to use GAs

After much experience in working with Genetic Algorithms we have found that the following is a simple yet effective representation of the process needed to use them:

1) Define a Problem
2) Define a Solution
3) Run the Solution

These phases are outlined in detail below.

Define the Problem

When presented with a problem to be solved it is first important to understand the freedoms and restrictions of the problem, and the expected outcome.

Take for example a simple problem:

Find X such that X^2 is the highest, with x being natural numbers between –2 and 5

From here we can infer that:

We want to search (or optimize - a premise for GA use)

The search is done on a number of X's,

The Xs are samples on a continuous dimension,

The dimension is bound between –2 and 5 and has 7 samples,

We are searching for a result of X^2,

We want to maximize the result.

For most cases, the following approach – implemented in our Genetic Algorithm workbench– covered the definition of most problems we were encountered with during a number of research activities.

1) Identify Data Types for the Problem
    a. Identify type – Continuous or categorical
    b. Identify data type values – classes or bounds & samples
2) Identify variables that belong to certain data types
3) Express a transformation that will express fitness in terms of those variables and other context information. This can be done by
    a. A generic function, script, program or object
    b. A set of rules of the form: *if [condition] then [delta$_T$] else [delta$_F$]*
4) Express the desire to maximize or minimize the fitness function.

An individual is then merely a representation of a solution. This representation is not done in terms of the problem variables but in terms of a problem independent alphabet. For ease of understanding in this paper we will assume that it is a binary alphabet (of 1's and 0's). For simplicity we will also assume the alphabet is known a priori and is constant. In the example above, an individual could be defined as three 1's and 0's: (-2=000, -1= 001, … , 4=110, 5=111).

Define Solution

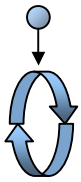
We call a "Solution" a set of parameters that will tell a Genetic Algorithm how to operate on a problem. Typical solution settings include:

1) A population size (we will assume single populations of fixed sizes)
2) Genetic operators, with associated activation probabilities and operator-specific parameters
3) Objective-Fitness transformation expressions, which will determine how a result in the objective function (e.g. X^2) maps to actual 'individual' fitness.
4) Cut conditions (which tell the GA runtime to stop the optimization process)

Run A Solution

Running a solution entails creating a population of individuals based on the problem definition and iteratively using process such as selection, operation and evaluation to create new populations of better individuals – until a cut condition is met. The actual processing of the run is done by a generic GA runtime.

The basic operation of the runtime is a loop that consists of generating new individuals based on fitness, and evaluating them. Operators cause genetic material to change as new individuals are created. The loop ends when a cut condition is met (Fig. 2)



Create a population, then…

(Fig.2)

Evaluate the new individuals of the population,

Based on fitness, select individuals to reproduce and create a new generation.

As they reproduce, apply operators that modify the genetic structure of the new individuals.

The three-step process and structure described above was implemented in a Genetic Algorithm workbench.

The GA Workbench

Using this simple yet flexible structure the workbench developed has been used to resolve problems such as:

"Solve a 100+ city blind traveling sales person (bTSP) problem"

"Train a neural network robot controller to find all lights in a maze – avoiding walls"

"Tune performance of any distributed computer application"

The workbench mainly consists of:

- A Windows user interface that lets users define problems, solutions, and control & monitor runs
- A runtime implemented as a series of COM objects which have the fundamental GA algorithms and some COM interfaces for user-defined data types, operators, cut conditions, user interface extensions etc
- A library of commonly used data types, operators, cut conditions, and fitness mappings.

This workbench saves problems as *.GAp* files and solutions as *.GAs* files. These are structured storage files that can be browsed using the data objects in the runtime.

The individual evaluation can be expressed as a VBScript function. This allows the user to enter a program with any desired complexity for the evaluation, even creating custom or 3rd party remote COM objects on other computers.

Using the GA Process and Workbench for a particular problem

1) Defining the problem
    a. Defining data types of the problem parameters
    b. Defining variables
    c. Defining an Evaluation of Performance
        i. Defining how to apply the configuration to a subject system

ii. Testing the simulation/system with the configuration

iii. Obtaining a 'performance' number

2) Defining a solution's parameters (population sizes, etc.)

3) Running the GA

Sample screen showing variables, their data type and their description, as entered for a particular application (Fig. 3)

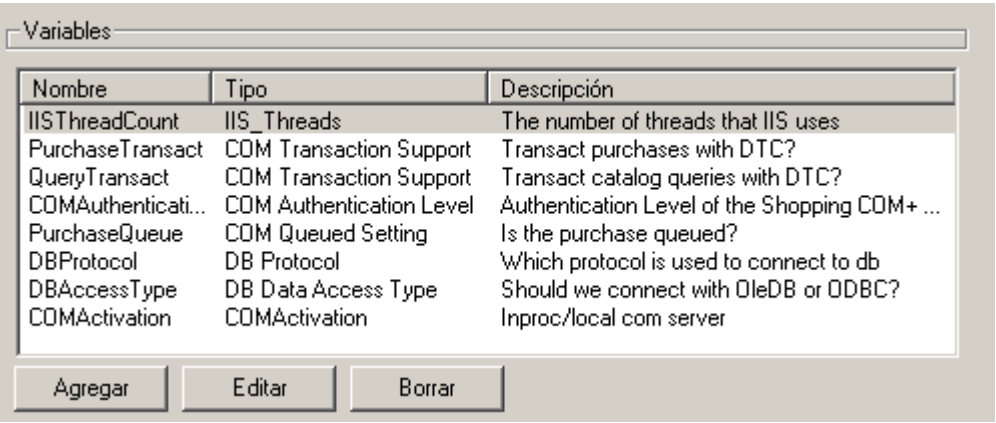
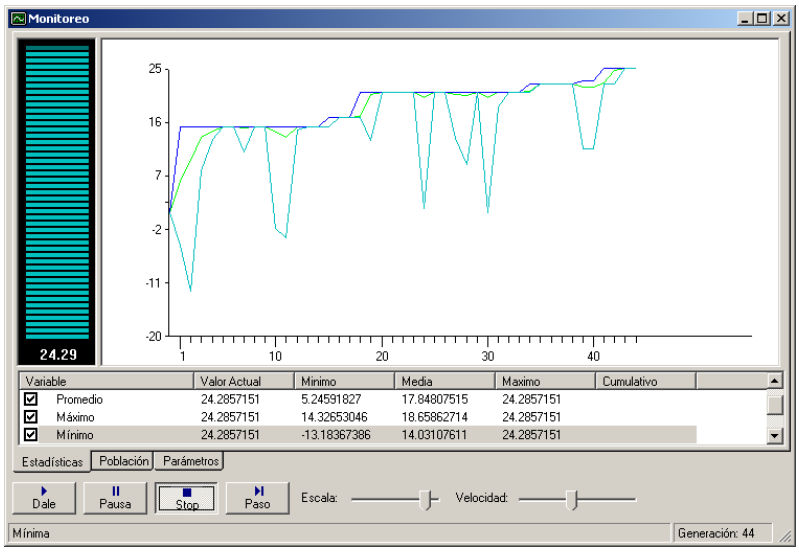


Figure 3: List of variables defined for the problem.

Sample screen where a user may enter his own problem to solve. The tool provides syntactic helpers so that the user does not need to master the programming language, and many parameters for the system are visually accessible.

This screen shows the monitoring of a genetic algorithm run, with increasing fitness values (Fig. 4)



(Fig. 4)

The image above shows the GA Workbench console during a sample run (1). The chart below displays maximum, minimum and average fitness for all individuals in the population. Note how the average tends to the maximum, and how the minimum can vary so much from the maximum, while the maximum is quite stable.

GAs and Archaeology

As observed, GA terminology usually has to do with rather outmoded mendelian ideas. We are fully aware of it. On the micro level, the scientific understanding of DNA structure has advanced a lot, and that terminology is no longer used. Today, genes and their behavior bear relatively little resemblance to simple parameter setting. Anyway, conventional GA and GP really work, and the new, more accurate metaphors (and their computational implementations) are still under construction.

GAs were occasionally used in social sciences. In 1979 R. G. Reynolds developed a GA-like adaptation schema to model prehistoric hunter-gatherer behavior. In 1981, Smith and DeJong used GA search to assess calibration of population migration. In 1985, R. Axelrod simulated behavioral norms using GA. The present paper is not an implementation of an archaeological problem, but an invitation to test a paradigm and the offering of a tool.

Why are we going to use GA in archaeology? Well, in the first place they are widely used in a multiplicity of domains, such as mathematics, engineering, political science, medicine, pattern recognition. After all, in non-analytic systems, semantics does not matter: algorithms are blind, and meaning-independent. There are relatively few problem structures, and they are the same across all disciplines. Besides, GA behaves especially well when processes are at stake, and archaeological problems generally are related to processual and diachronical issues. We think GA will be helpful to understand the underlying structure of an archaeological problem as a generic problem:

- Does your archaeological reasoning really qualify as a problem?
- Do you know enough about your subject matter in order to formulate a problem, or something essential is missing?
- How good a problem is it? Is it trivial? Is it solvable?
- What does a GA implementation of your archaeological problem look like?

GAs are specially suitable for problems having nonlinear search spaces and highly polymodal problems. They help to asses the solution process as a search procedure, and today we have a deep knowledge of them. Furthermore, they help to automate the invention process, which involves factors such as intuition, creativity, abduction. Once upon a time, anthropology and archaeology developed the first evolutionary theories and concepts outside biology. With GA, in a sense, evolution theory comes back home.

Notes.

1-The image is provided for illustrative purposes. Screenshots were not obtained in the actual experiment.

Further Reading

- Axelrod, R., 1985. *Modeling the evolution of norms*. Paper presented at the American Political Science Association Meeting, New Orleans.
- Holland, J., 1975. *Adaptation in natural and artificial systems*. Ann Arbor: The University of Michigan Press.
- Goldberg, D., 1989. *Genetic algorithms in search, optimization & machine learning*. Addison Wesley, Reading.
- Koza, J., 1992. *Genetic Programming: On the programming of computers by means of natural selection*. Cambridge: MIT Press.
- Koza, J. et al, 1999. *Searching for the impossible using genetic programming*. Proceedings of the Genetic and Evolutionary Computation Conference, Orlando. San Francisco: Morgan Kaufmann.
- Reynolds, R. G., 1979. *An adaptive computer model for the evolution of plant collecting and early agriculture for hunter-gatherers in the valley of Oaxaca, Mexico*. Unpublished doctoral dissertation. Ann Arbor: University of Michigan Press.
- Smith, T. and K. A. DeJong, 1981. Genetic algorithms applied to the calibration of information driven models of US migration patterns. *Proceedings of the 12th Annual Pittsburgh Conference on Modeling and Simulation*, pp. 955-959.

Carlos Reynoso is Professor in Anthropology at the University of Buenos Aires. He is author of several books on anthropologist theory. He was Senior Technical Advisor in Microsoft.

Edward Jezierski is System Ingeneer and Technical Specialist Researcher in Microsoft.