DESIGN AND IMPLEMENTATION OF AN INTERFACE FOR
A FUNCTION OPTIMIZATION TOOL
BASED ON GENETIC SEARCH TECHNIQUES

H. Matiaudan, A. Paccanaro, H. Soley

Laboratorio de Electrónica Digital
Universidad Católica "Nuestra Señora de la de Asunción"
Asunción, Paraguay
Tel: +595-21-334650
Fax: +595-21-310587
e-mail: apaccana@ledip.py

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano
Milano, Italy

ABSTRACT

This paper presents the design and the implementation of a user-friendly graphical interface for GENESIS (GENetic Search Implementation System), a well known system for function optimization based on genetic search techniques, developed by John Grefenstette. The user-friendly interface was carefully studied through a long monitoring of the use of GENESIS and it was implemented maintaining all the features of the original version. It has proven to be very helpful both in teaching Genetic Algorithms and to ease the use of the application for research purposes.
In the sequel, an informal introduction to the basic concepts and terminology of Genetic Algorithms is given, followed by a presentation of the interface.
1. AN INFORMAL INTRODUCTION TO GENETIC ALGORITHMS

Genetic algorithms appear to hold a lot of promise as general purpose adaptive search procedures. The basic concepts of GA's were developed by John Holland (Holland [1975]) and his students (Betchek [1981], DeJong [1975], Frantz [1972], Holstein [1971]) at the University of Michigan. Theoretical considerations show that genetic techniques provide a highly efficient heuristic for information gathering in complex search spaces. In practice, there are a great number of experimental studies which have shown that GAs exhibit impressive efficiency. In general, it is possible to affirm that, while classical gradient search techniques are more efficient for problems which satisfy tight constraints (e.g., continuity, low dimensionality, unimodality, etc.), GAs consistently outperforms both gradient techniques and various forms of random search on more difficult (and more common) problems, such as optimizations involving discontinuous, noisy, high dimensional, and multimodal objective functions. GAs have been applied to various domains, including numerical function optimization (Brindle [1981]), adaptive control system design (DeJong [1980]), and artificial intelligence task domains (Smith [1983]).

GAs belong to the class of those algorithms based on the principle of evolution (survival of the fittest), where a population of individuals (potential solutions) undergoes a sequence of unary (mutation type) and higher order (crossover type) transformations. These individuals strive for survival: a selection scheme, biased towards fitter individuals, selects the next generation. After some number of generations, the program converges - the best individual hopefully represents the optimum solution.

In the following of this paragraph a brief, informal description of how GAs work is given in order to present the basic concepts and terminology that will be used in the second part of this paper.

GAs are iterative procedures which maintain a "population" of candidate solutions to the objective function f(x) which has to be optimized. This population evolves in time. Let us write:

\[ P(t) = \langle x_1(t), x_2(t), \ldots, x_n(t) \rangle \]

to indicate the population at the time t constituted by the candidate solutions (or individuals) \( x_1 \ldots x_n \).

Each individual \( x_i \) in this population is simply a binary string of length \( L \); the semantics associated with it is unknown to the GA. During each iteration step, called a "generation", the current population is evaluated and on the basis of that evaluation a new population of candidate solutions is created.

At the first iteration the initial population \( P(0) \) is chosen at random or may contain heuristically chosen initial points. Then each structure in \( P(0) \) is evaluated. Evaluation consists in calculating the value of the objective function for each individual in the population.

In order to generate the individuals of the population \( P(t+1) \), first structures are chosen from the population \( P(t) \) by a randomized "selection procedure" that ensures that the expected number of times a structure is chosen is proportional to that structure's performance, relative to the rest of the
population. At the end of this process, population P(t+1) will therefore contain only exact duplicates of structures of the population P(t).

In order to search other points in the search space - possibly better solution to the problem - some variations must be introduced into the new population. This is done by means of idealized "genetic recombination operators." The most important recombination operator is called "crossover". Under the crossover operator, two structures in the new population exchange portions of their binary representations. This is done by choosing a point at random, called the crossover point, and exchanging the segments to the right of this point. For example, let

$$x_1 = 100:01010, \text{ and } x_2 = 010:10100.$$  

and suppose that the crossover point has been chosen after the third digit (as indicated by a ":" ). The structures resulting from the crossover operation would be

$$y_1 = 100:01100 \quad \text{ and } \quad y_2 = 010:01010.$$  

There are 2 main theoretical reasons for crossover, which may be informally explained as follows.

Crossover serves two complementary search functions. First, it provides new points for further testing within the schemes already present in the population. A schema is a string over an extended alphabet (0,1,#) where the 0 and the 1 retain their normal meaning, and the # is a wild card or don't care symbol. In the above example, both $x_1$ and $y_1$ are representatives of the schema 100####, where the # means "don't care". Thus, by evaluating $y_1$, the GA gathers further information about this schema.

The second theoretical reason of crossover is that it introduces representatives of new schemes into the population. In the above example, $y_2$ is a representative of the schema #1001###, which is not represented by either "parent". If this schema represents a high-performance area of the search space, the evaluation of $y_2$ will lead to further exploration in this part of the search space.

Finally the iteration process typically terminates when an acceptable approximate solution to $f(x)$, is found, or when a certain number of generations has been reached.

2. RUNNING GENESIS VERSION 5.0.

GENESIS (GENetic Search Implementation System) version 5.0 was developed by John Grefenstette (Grefenstette [1984a] [1984b]) and it is a well known system for function optimization based on genetic search techniques. In this paragraph a short overview of how the original version of the application works is presented:
GENESIS in its original version, is a program designed to run on various architectures in a portable manner. Program operation is specified by feeding parameter files; file suffix specifications are used to differentiate one project from another. This information was stored in a "in.xxx" and "template.xxx" files. Some of the parameters to be introduced where:

-- the suffix for file names for the project
-- genes specifications
Each gene takes on a range of floating point values, with a user-defined granularity and output format. The user was asked to specify for each gene: its minimum value; its maximum value; the number of values (a positive power of 2); the desired output format for this gene (using printf format, e.g., %7.2f). The user had also the possibility to specify a repetition count, meaning that there were a number of genes with the same range, granularity, and output format. When all genes had been specified, this information was stored in the "template.xxx" file.

-- the number of experiments, i.e. the number of independent optimizations of the same function.
-- the number of trials per experiment
-- the population size
-- the length of the structures in bits
-- the crossover rate
-- the mutation rate
-- the generation gap, i.e. the fraction of the population which is replaced in each generation.
-- the seed for the random number generator [123456789]:
-- the running options
Setting the running options was rather complicated, since the user had to know the meaning of each one of them from the user manual. Some of these were:

"a": evaluate all structures in each generation.
"b": at the end of the experiments, write the average best value (taken over all experiments) to the standard output.
"c": collect statistics concerning the convergence of the algorithm.
"C": collect statistics concerning the performance of the algorithm.
"d": dump the current population to "ckpt.xxx" file after each evaluation.
"D": display mode. Performance statistics were printed to the screen after each generation.

Once the setup program terminated, the user had to write, using a Text Editor, the evaluation function. This would take one structure as input and returns a double precision value. The procedure had to be declared as follows:

```c
double eval(str, length, vect, genes)
char str[];
int length;
double vect[];
int genes;
```

where "str" is string representation of the structure, "length" is the length of "str", "vect" is the floating point representation, and "genes" is the length of "vect". The body of the evaluation function was, of course, application dependent. Figure 1 shows a sample evaluation function (a parabola).
double eval(str, length, vect, genes)
char str[];       /* string representation */
int length;     /* length of bit string */
double vect[];   /* floating point representation */
int genes;       /* number of elements in vect */
{
    register int i;
    double sum;

    sum = 0.0;
    for (i = 0; i < genes; i++)
        sum += vect[i] * vect[i];

    return (sum);
}

Figure 1

After executing the Setup program and editing the evaluation function the user
had to compile the function and link it to the application. Finally GENESIS
could be executed. Results were stored in the file report.xxx, report.xxx and
log.xxx. To see them this files had to be opened using a text editor. GENESIS
did not have any on line help; a text file of the application manual was
available.

3. DESIGN AND IMPLEMENTATION OF A USER-FRIENDLY INTERFACE FOR GENESIS.

This way of working was rather harsh to the user. Moreover it limited the use
of the application for didactic purposes. A front-end to the program was
therefore developed, that includes an on-line help and an intuitive handling
of the project files. The idea was to obtain a system which would be easier to
use for research purposes and that at the same time could be used in a class-
room for teaching Genetic Algorithms.

To design the interface, the typical work session with GENESIS was observed.
Moreover user were asked to express their ideas on how to improve the program
utilization. Based on these considerations an interface was designed and im-
plemented, and is presented in the following.

All GENESIS functionalities were maintained moreover the application keeps
using the same files as before, in order to maintain the possibility of execut-
ing the program on various architectures and to allow future integration
with program upgrades.

The graphical interface was built for the Windows environment using C++ object
oriented programming language. The compiler used was the Borland C++ for Win-
dows which allow recompilation for Windows 3.1, Windows NT, and DOS. In de-
signing it the guidelines for writing a standard Windows application were
followed (Petzold [1992]). This project was entirely developed at the LED, La-
boratorio de Electronica Digital, of the Catholic University of Asunción,
Paraguay.

The new interface allows the user to run the GENESIS application without the
need of any other tool. The interface constitutes an integrated environment
containing: a text editor (for writing the evaluation function), commands for
compiling and executing GENESIS and an on-line help. The need of the Setup program has been substituted with dialog boxes which are used to enter all the input parameters, including genes characteristics and run options. Results are visualized in text windows.

The application works using projects, each one of them being constituted of various files. As stated before these files are same which were used by the original version of GENESIS (in.xxx, template.xxx, out.xxx, ...). This allows the possibility of eventually porting these files in other environments (UNIX, DOS) and then execute the application. The advantage is that now the user does not need to edit them directly or execute the Setup program each time that they have to be modified.

Let us analyze the functionalities of the interface examining its most important menu items: FILE, PROJECT and RUN.

The FILE menu contains all the commands referring to storing and opening projects, such as: New Project, Open Project, Save Project, Save As and Exit.

The PROJECT menu contains all the commands referring to modify each one of the elements constituting the projects. Particularly the following items are present: Edit Evaluation Function, Setup Genesis, Gene Setup.

Edit Evaluation Function: this command is used to edit the evaluation function. Selecting it causes a text window to be opened which already contains the function declaration and its parameters. This window is shown in fig. 2

![Image of the interface showing the F1.C - Function and the myproject.gpr file.](image)

Figure 2

Setup Genesis: this command activates a dialog box where the user specifies the system configuration. This dialog box is shown in fig. 3
Gene Setup: this option activates a dialog box where the parameters related to the genes have to be fed in. Arrows buttons (left and right) are used see and edit the various in the order in which they have been entered. This dialog box is shown in fig. 4
Finally the RUN menu contains all the commands needed to set up all the execution options and compiling the applications. Particularly the following items are present: Program Options, Compile, Run Genesis. Let us analyze each one of them:

Program Options: this command causes a dialog box to appear where the user specifies execution options. This dialog box is shown in fig. 5

![Genetic Algorithm Options](image)

Compile: this command will cause the evaluation function entered by the user to be compiled and linked with the other GENESIS files with any C language compiler available (the type and location of the C compiler are specified when installing the program).

Run Genesis: this command will cause GENESIS to be executed, using the configuration and options previously described. When the execution terminates a report window containing the results will be visualized. These results will also be stored in a "report.xxx" file.
An example of the report window is shown in fig. 6
4. CONCLUSIONS AND FURTHER DEVELOPMENTS

The purpose of this project was to create an application for function optimization based on genetic search techniques that would be easy to use for research purpose and at the same time useful for teaching genetic algorithm. An interface was built for GENESIS version 5.0, a powerful and well known application which was lacking of a user friendly interface. It was developed for the Windows environment. Among its main features are an intuitive way of managing projects, setting up run parameters and controlling execution. Compatibility of the kernel with other system architecture has been preserved. The system has been used in the classroom during courses of the last year of the career of Computer Science Engineering at the Catholic University of Asunción, Paraguay, and has given good result in helping students to grasp the concepts of Genetic Algorithm. It is also currently used for research purposes.

We have a project to extend this interface. The idea is to allow the user to give a more insight look of how GAs work. We would like to provide a tool which would show all the details of the various operations - we believe this would be very interesting for didactic purposes. We are thinking, for example, to a graphic way for representing the evolution of populations and for allowing the user to see and control the effect of the various operators on the elements of the populations.
REFERENCES


A. Brindle[1981], Genetic algorithms for function optimization, Ph. D. Thesis, Computer Science Dept., Univ. of Alberta.


J. J. Grefenstette[1984b], A user's guide to GENESIS. Technical report No. CS-84-11 Nashville, Vanderbilt University, Department of Computer Science.


