AN XML-BASED GENETIC ALGORITHM FOR NONLINEAR OPTIMIZATION PROBLEMS

Dep. of Civil and Env. Eng., Ehime University ○ Amílcar Soares Júnior (Research Student)
Dep. of Civil and Env. Eng., Ehime University Camilo Allyson Simões de Farias (Dr. of Eng., Student)
Dep. of Civil and Env. Eng., Federal University of Paraíba Celso Augusto G. Santos (Dr. of Eng., Assoc. Professor)
Dep. of Civil and Env. Eng., Ehime University Koichi Suzuki (Dr. of Eng., Professor)

1. INTRODUCTION

Optimization is a common problem in many fields of science. The optimization of a mathematic function corresponds to the search of its maximum or minimum value. Many techniques have been proposed in order to find these values. However, most of these traditional techniques are not very efficient for solving nonlinear functions. Evolutionary Algorithms (EAs), which are defined as a set of probabilistic optimization methods based on the theory of evolution of Charles Darwin[1], appears to handle arbitrary types of problem objectives and constraints. The species behavior observed by Darwin is simulated computationally in order to obtain optimized values for the parameters. One of the implementations of EAs, called Genetic Algorithms (GAs), was developed by John Holland[2] in the end of the 1960s. Since this first work, many GA implementations have been developed but most of them were built to a specific purpose.

This work aims at presenting the development of a GA tool that can be used for solving any type of optimization problem. A portable, configurable and general purpose GA software was constructed utilizing the Java (*Sun Microsystems*) programming language and the eXtended Markup Language (XML).

2. THE GENETIC ALGORITHM

As mentioned above, the GAs are inspired in the natural evolution process. It means that as better as an individual fits in the environment, greater is the possibility of it survives and generates *offsprings*. A set of solutions are called *population*. Individuals called *chromosomes* compose a population. Each chromosome represents a possible solution to the problem. Each solution is evaluated and generates a *fitness*.

As can be seen in Figure 1, a GA must initially generate a population of individuals. The first population of individuals is then analyzed. The fitness for each individual is determined, i.e., how much the proposed solution is relatively better compared to the other individuals. Like the theory of Darwin, the strongest individuals are more likely to be selected and less inclined to be discarded. While the stopping criterion is not reached, the algorithm creates new populations of individuals. Through the *alter population* operation, techniques such as crossover and mutation are applied in the selected individuals of the population. These techniques alter the values of the parameters and are the key concepts that make the GAs a powerful optimization technique. In the crossover operation, as in the nature, parts of the number (generally bits) are exchanged between two individuals generating an *offspring* different from the originals. In the mutation operation each bit of the offspring has a small probability to be flipped. It means that if the mutation occurs in one bit, a 0 becomes a 1 and vice versa.

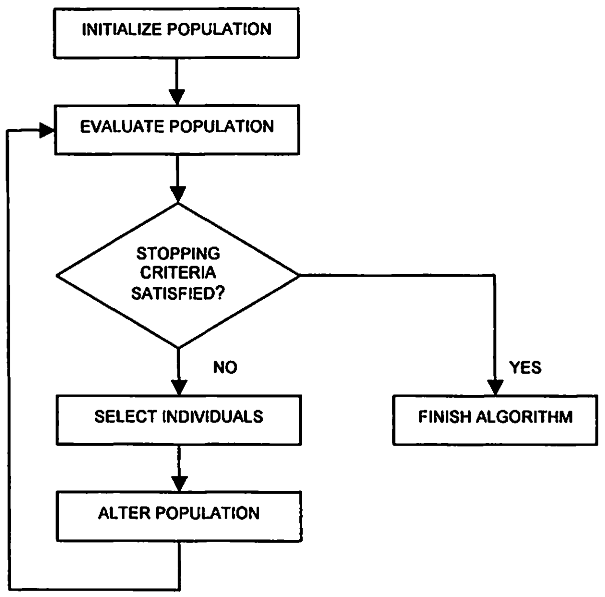

Figure 1 The GA general description.

3. HOW THE SYSTEM WORKS

Nowadays, the use of GAs as a method for optimizing functions is very common in engineering[3]. However, most of the GAs implementations are applied to specific cases. This proposal suggests that the application's source code might be not changed, so there is only one configuration file to be previously filled by the user.

Figure 2 shows how the system works. The user selects the desired options in order to perform the optimization (parameters, objective function, restrictions, crossover rate, etc.). The system loads the information from the file created by the user, runs the optimization and creates a file with the output values.

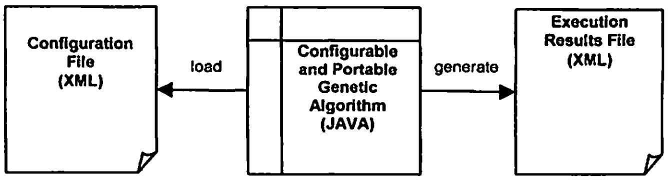
Configuration File (XML)
load
Configurable and Portable Genetic Algorithm (JAVA)
generate
Execution Results File (XML)

Figure 2 How the system works

4. UTILIZED TOOLS

In order to build a portable GA, it was necessary the adoption of a programming language that allows a multiplatform execution. The Java programming language was used because it is an object-oriented language different from the conventional ones. This language uses the *bytecodes* (intermediary language) approach. This approach allows the portability of any application platform developed in such language, which is independent of the Operational System that is being used. As a consequence, the developed tool can run in different Operational Systems such as Windows, Linux, etc.

The eXtensible Markup Language (XML) was utilized so as to describe the GA simulation (input and output files) and to represent a flexible and extensible data[4)]. The elements and attributes described in this language provide information about the data. XML formatted data can be manipulated by different applications and modified in accordance with the needs that are emerging. The main goal of using it to describe the GA simulation is that the output data can be used by other software with the purpose of building more complex tools.

5. RESULTS

The GA implementation was applied so as to find the minimum value of the Goldstein-price function. This nonlinear function is described by Equation (1) and shown in Figure 3.

Equation (1):

$$f(x,y) = (1+(x+y+1)^2(19-14x+3x^2-14y+6xy+3y^2)).$$
$$(30+(2x-3y)^2(18-32x+12x^2+48y-36xy+27y^2))$$

subject to

$$-2 \le x \le 2$$
$$-2 \le y \le 2$$

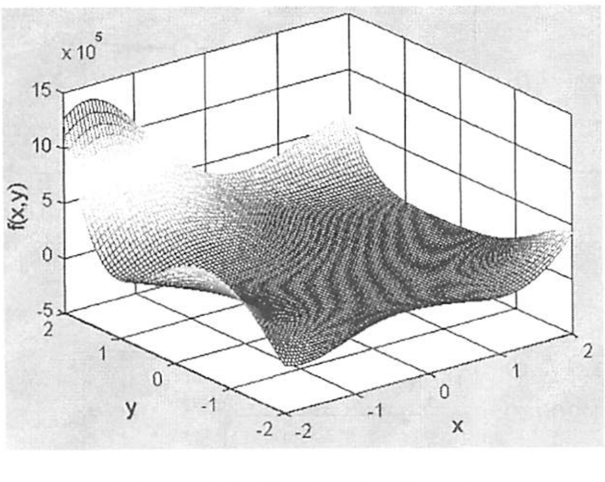

Figure 3 Goldstein-price function

As can be seen in Figure 3, the minimization of this function is not an easy task. The optimal minimum value, which is equal to 3.0, is located at position (x,y) = (0,-1.0).

In the present simulation each generation was composed of 60 individuals. At the beginning a grid was set and the individuals were equally distributed along the limits for x and y. The evolution of the algorithm from the initial population until the 60th generation is illustrated in Figures 4 and 5. Examination of the results shows that most of the solutions converged to the optimal value by the 60th generation. The stopping criterion was reached at the 179th generation and the exact optimum was found.

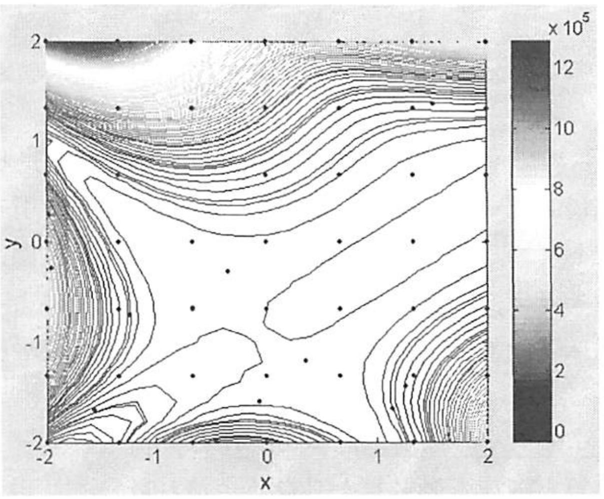

Figure 4 Initial population

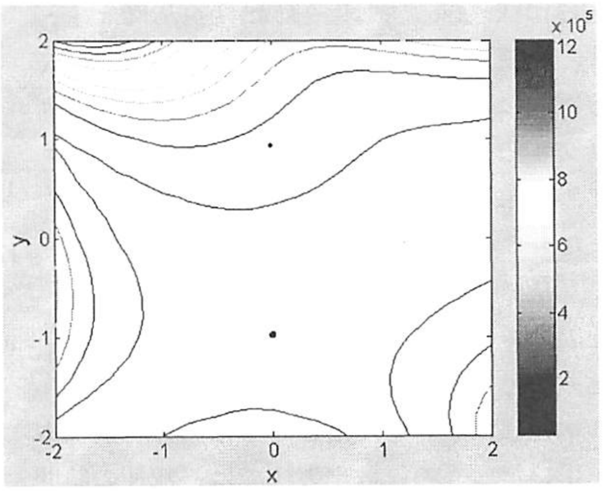

Figure 5 Population at the 60th generation

6. CONCLUSIONS

Many GAs have been developed with the intention of solving specific problems. The main goal of the developed tool, which is implemented using Java and XML, was to allow the user to solve any type of optimization problem. Once properly configured, the tool can optimize mathematical functions or more complex applications. As a result, this GA implementation may be very useful as a support for various fields of science.

REFERENCES

1) Beyer, H.G., (2001). "The Theory of Evolution Strategies". *Springer, Inc.*, 380 p.

2) Holland, J. H., (1975). "Adaptation in Natural and Artificial Systems", *MIT Press.*

3) Santos, C.A.G., Srinivasan, V.S., Suzuki, K. & Watanabe, M. (2003) "Application of an optimization technique to a physically based erosion model". *Hydrol. Processes* **17**, 989–1003, doi: 10.1002/hyp.1176.

4) Graves, M. (2003) "Projeto de Banco de Dados com XML", *Makron Books, Inc*, 518pp.