JGA User Guide

WinNT/2000 Deployment
v.2005.06.20

Andrés Medaglia (amedagli@uniandes.edu.co)
Centro para la Optimización y Probabilidad Aplicada
Departamento de Ingeniería Industrial
Universidad de los Andes
http://copa.uniandes.edu.co

Eliécer Gutiérrez G (egutierr@uniandes.edu.co)
Centro para la Optimización y Probabilidad Aplicada
Departamento de Ingeniería Industrial
Universidad de los Andes

Created: September 20, 2004
Updated: June 20, 2005

Table of Contents

WHAT IS JGA?

JGA is a java evolutionary optimization engine useful for solving optimization combinatorial problems. Its name is an acronym for **J**ava **G**enetic **A**lgorithms.

A Genetic Algorithm (GA) is a directed random search technique developed by Holland (1975), which can find the global optimal solution (or very near solutions) in complex multi-dimensional search spaces. A GA is based on natural evolution in that the operators it employs are inspired by the natural evolution process. These operators, known as genetic operators, manipulate individuals in a population over several generations to improve their fitness gradually. GA foundations are described by Pham (2000), Rayward (1996) and Michalewicz (1996). GAs and their applications are described in detail by Gen (2000).

LICENSE

You are free to use JGA for academic or research use. For commercial use, please contact the authors first for approval. If you use JGA in your research, please give proper credit to the authors.

REQUIREMENTS

The requirements for JGA are:

- Java 2 Platform, Standard Edition (J2SE)
  - J2SE 1.4.x or above
  - Free and available at http://java.sun.com/j2se/index.jsp
- IDE for Java
  - Eclipse (recommended) is a kind of universal tool platform - an open extensible IDE for anything and nothing in particular.
  - Free and available at http://www.eclipse.org
  - JGA has been designed to run without Eclipse. However, this IDE facilitates enormously the development and execution of JGA applications.
- Operating system
  - Any platform in which Java runs (e.g., Windows, Linux, Solaris)
- Memory
  - 128 MB (minimum)
  - 256MB RAM (recommended)

FILES IN DISTRIBUTION

After expanding the distribution file `jga-YYYYMMDD.zip`[1] in the target directory, you will have the folder structure described on Table 1.

Table 1: Folder structure

Folder	Description	Typical Value
`[target]`	Target folder. Throughout this document we use the alias of `[target]` for this folder.	`c:\jga`
`[target]\lib`	Folder with libraries (jars) needed to use JGA. • jga-YYYMMDD.jar: JGA library. • ioutils-YYYMMDD.jar: libraries for common input-output tasks.	`c:\jga\lib`
`[target]\doc`	Folder with java documentation.	`c:\jga\doc`
`[target]\man`	Folder with documentation such as the user guide and presentations.	`c:\jga\man`
`[target]\examples`	Folder with examples that illustrate the use of JGA.	`c:\jga\examples`

[1] Where YYYY stands for the year, MM stands for the month, and DD stands for the day of the version of JGA.

How to Run a JGA Application via Eclipse: Step By Step

This section is useful for two reasons: 1) it helps you set up a project in Eclipse and 2) it tests your installation by running an example provided in the JGA distribution.

1. Launch Eclipse.

   You should see the screen shown in Figure 1. Note that the red-dotted area is particular to every user. For instance, those are the current Eclipse projects for a given user.

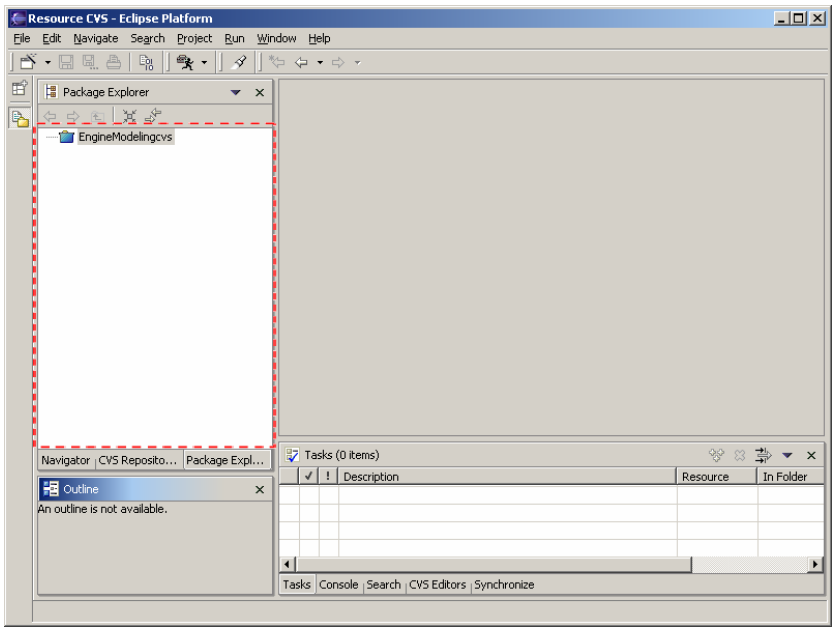

Figure 1: Eclipse first screen

2. Create a new project called myJGAApp for the Ackley example.
   a. Click on the icon shown in Figure 2.

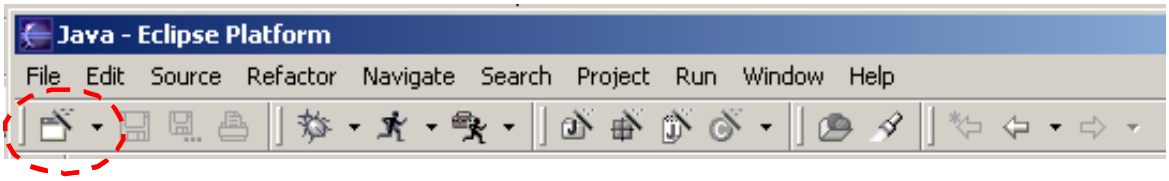

Figure 2: File-new

b. Select Java Project and click on the Next button in the dialog shown in Figure 3.

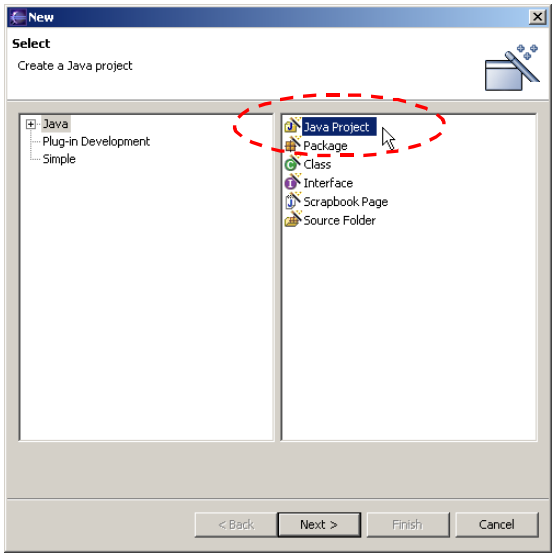

Figure 3: Creating a new project

c. Give the project the name of myJGAApp and point to the folder for the Ackley example as shown by the red-dotted line in Figure 4. Click Next.

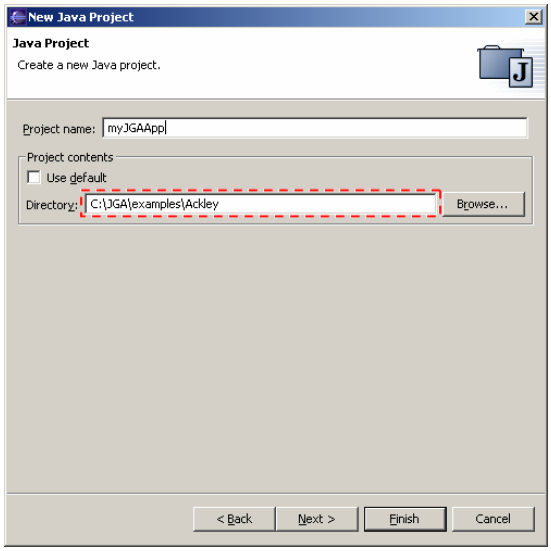

Figure 4: Pointing to the example folder

d. Figure 5 shows the Source tab on the Java Settings dialog that defines the location of your source code. In this case, the code for the Ackley example resides at `[target]\examples\Ackley\src`. Click on Finish.

The project myJGAApp has been successfully created! You will see the project been added to the left side of the panel on Eclipse as shown in Figure 6 (see the myJGAApp project).

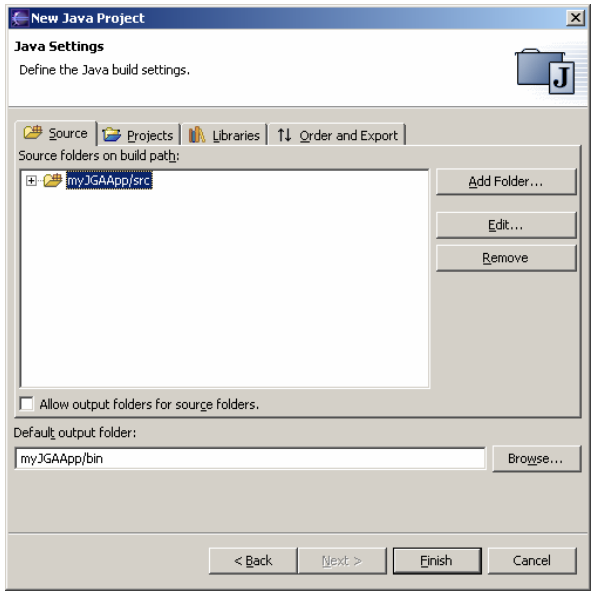

Figure 5: Java Settings dialog: Source tab

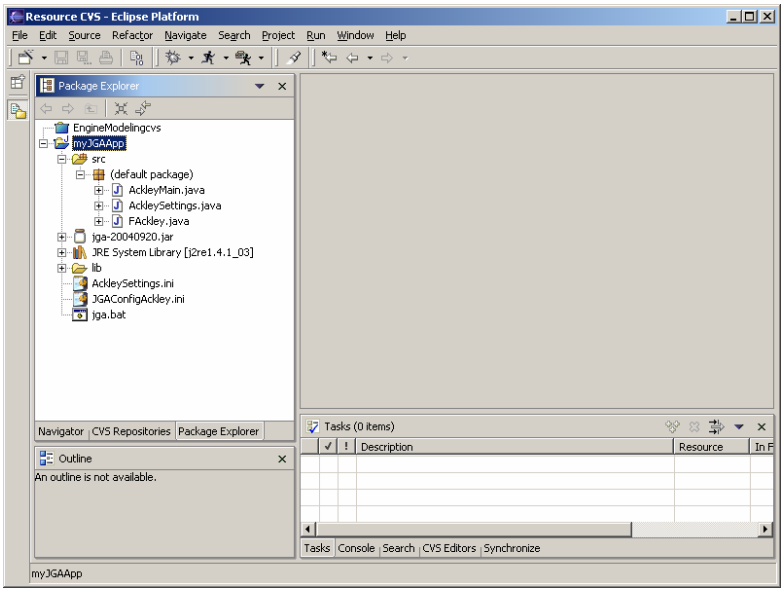

Figure 6: myJGAApp project

You can change the visualization format from package explorer to navigator by clicking on the labeled tab or bu using the window-show view menu.

e. Figure 7 shows the expected result if you click on the Libraries tab on the Project properties dialog. Note that the libraries jga-20050626.jar and ioutils-20050626.jar of compiled Java code have been automatically picked from `[target]\examples\Ackley\lib`.

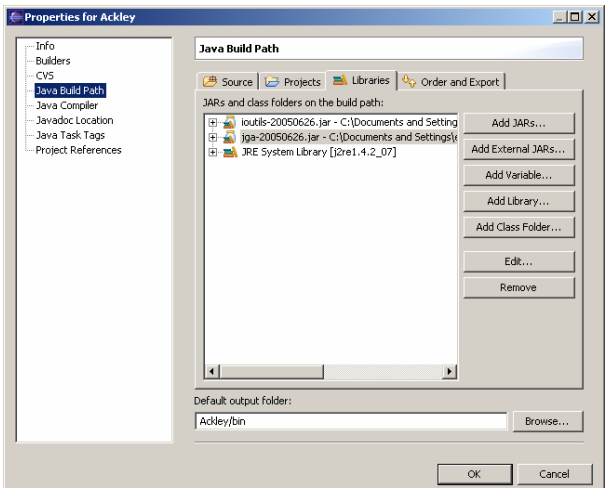

Figure 7: Java Settings dialog: Libraries tab

If the libraries do not appear, you can link the libraries to the project by clicking on the Add External JARs buttons and selecting the jars files from `[target]\examples\Ackley\lib`

3. Compile and run the Ackley example.
   a. Select and Click on Project - Rebuilt All menu as shown in Figure 8 to compile the source files.

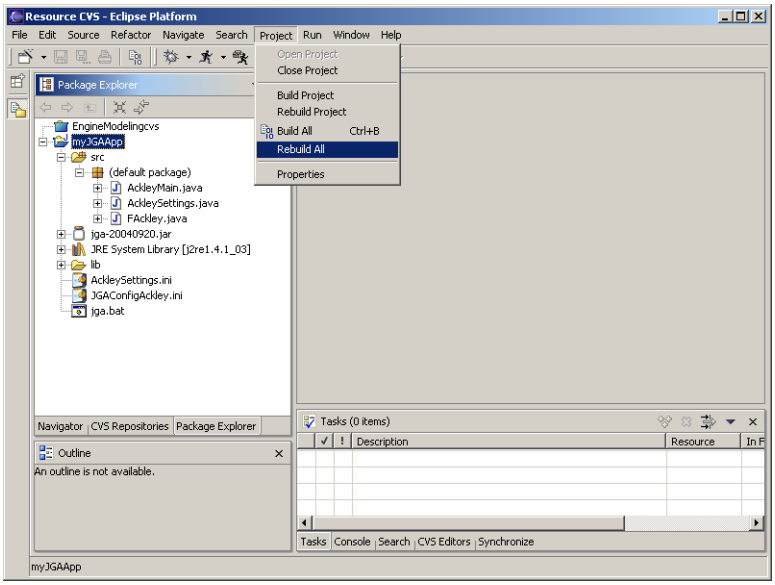

Figure 8: Running an example

b. Select and Click on the Run menu as shown in Figure 9 to run the application the first time.

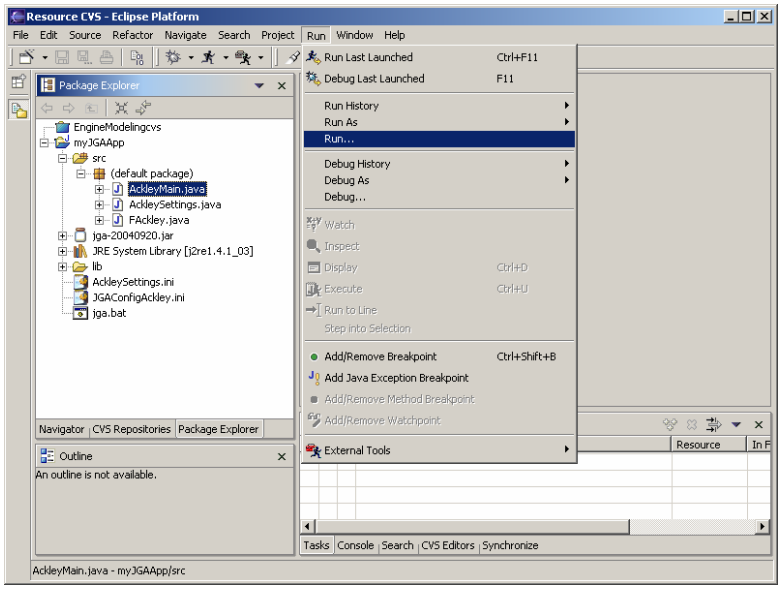

Figure 9: Running an example

c. Click on the Java Application on the left panel (labeled Configurations) and the Main tab as shown in Figure 10.

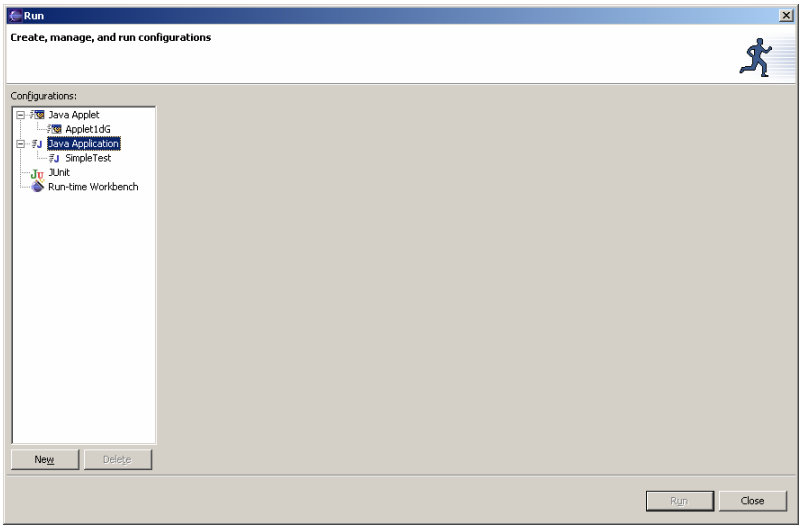

Figure 10: Selecting Java Application

d. Click on the New button on the botton panel on the Run dialog. The run configuration information will be appearing as shown in Figure 11 .

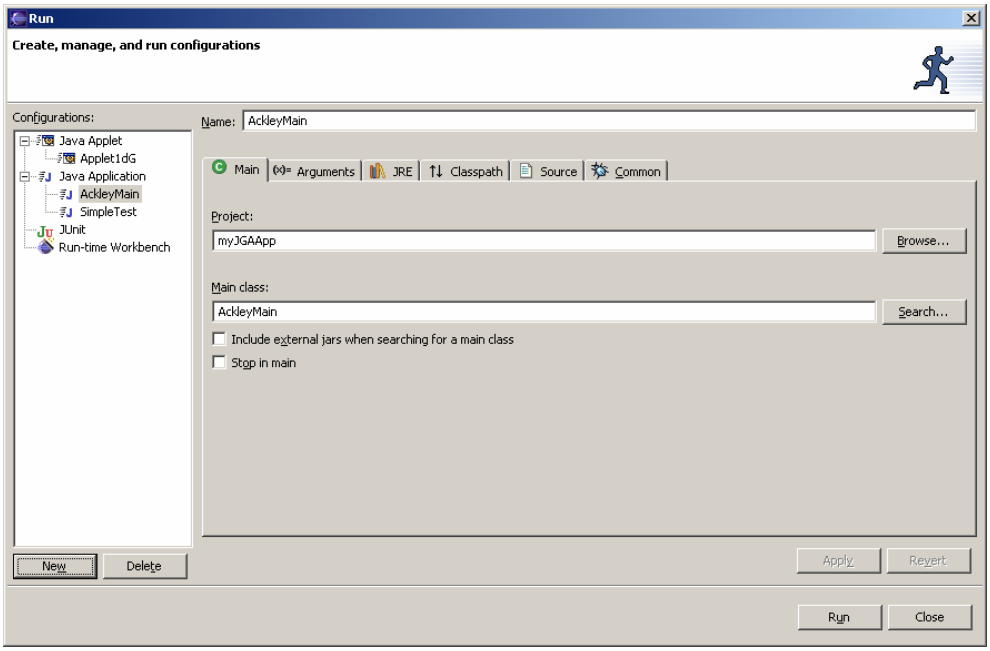

Figure 11: Configuring the Main tab on the Run dialog

e. Click on the (x) Arguments tab on the Run dialog and fill in the information as shown in Figure 12. Type the name of the configuration file, namely `JGAConfigAckley.ini`, which resides in the folder `[target]\examples\Ackley`.

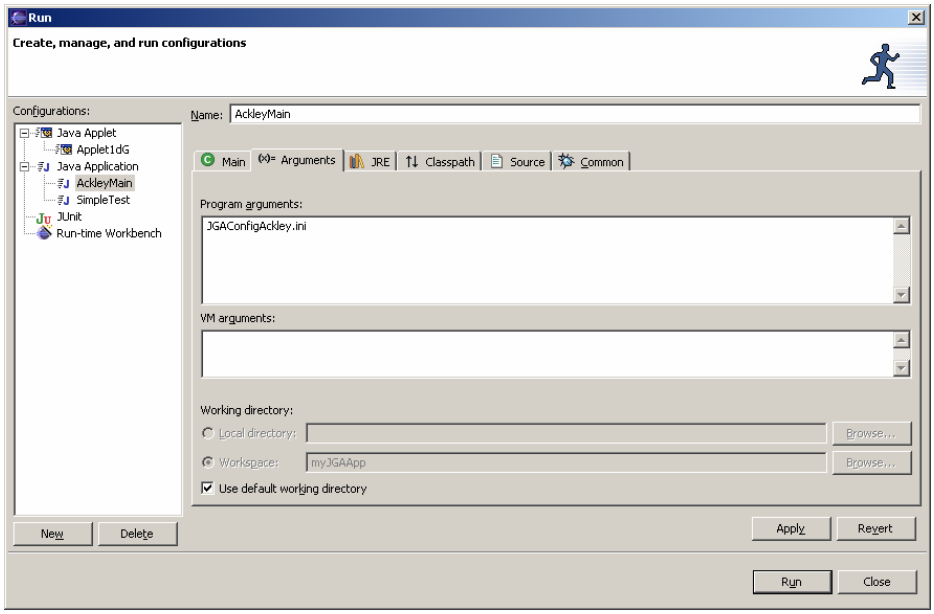

Figure 12: Configuring the (x) Arguments tab on the Run dialog

f. Click on Apply and Run. Figure 13 shows the Console window that displays the trace of the execution as soon as the example starts running.

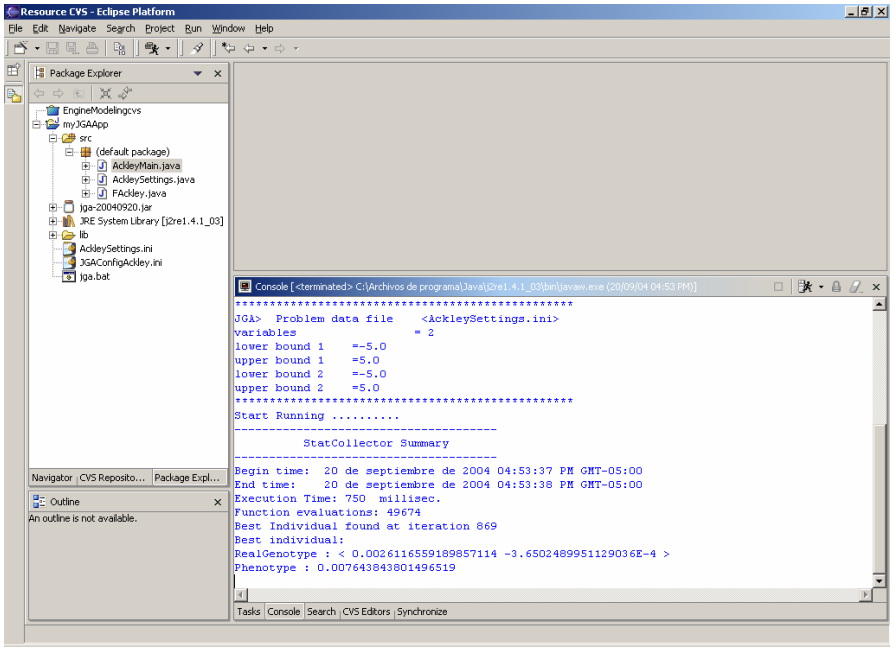

Figure 13: Running an example: console window

To run the application the next time you can use directly the button 

CODING A GA APPLICATION FOR SOLVING A PROBLEM

The Ackley problem

The Ackley Problem (Ackley,1987) is a minimization problem. Originally this problem was defined for two dimensions, but the problem has been generalized to *N* dimensions. Formally, this problem can be described as finding a vector, that minimizes the following equation, named The Ackley function:

$$\min f(\mathbf{x}) = a + \exp -a \cdot \exp^{-b\sqrt{\frac{1}{N}\sum_{j=1}^{N} x_j}} - \exp^{\frac{1}{N}\sum_{j=1}^{N}\cos(2\pi \cdot x_j)}$$

The optimum solution of the problem is the vector v = (0,...,0) with F(v) = 0. A plot for the Ackley function in two dimension is shown in Figure 14.

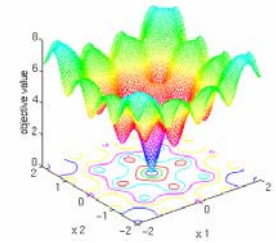

Figure 14: The Ackley function.

Understanding the JGA architecture

At first, it is necessary to understand the global architecture for a GA application using JGA. Figure 15 shows this architecture.

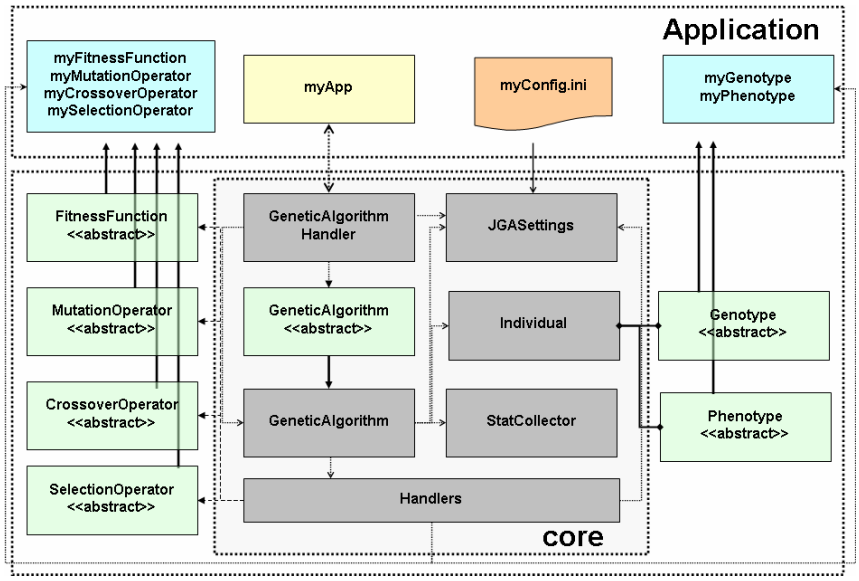

Figure 15: Architecture for a JGA application

The JGA package consists of a group of concrete classes which defines the core of the package. Additionally, there are a group of abstract classes to allow users to define their own problem specific classes. The user application classes are enclosed with a dotted box in the upper part of the diagram.

The user must construct or select from the built-in classes the classes required for solving the specific problem. Also, he must create or edit a text configuration file to set the parameters for the genetic algorithm. If the user constructs their own classes, these classes must be extended from abstract classes defined in the library.

The components which the user must construct or select from built-in classes are:

- The genotype
- The phenotype
- The mutation operator
- The crossover operator
- The selection operator
- The fitness function
- The genetic algorithm procedure

For all the components, except for the fitness function, the library provides a set of standard built-in classes to make easy the implementation of new applications. Table 2 shows the java components used to implement the GA for The Ackley problem. You can edit to review the java source code of the user-defined classes in the eclipse project.

Table 2: Summary of JGA classes used for the Ackley problem

Component	Class	Extended from Class[1]
Genotype	IntegerGenotype ♦	*Genotype* ♦
Phenotype	SingleFitnessPhenotype ♦	*Phenotype* ♦
Fitness function	FAckley •	*FitnessFunction* ♦
Mutation operator	RandomAssigmentRealMutation ♦	*MutationOperator* ♦
Crossover operator	SinglePointRealCrossover♦	*CrossoverOperator* ♦
Selection operator	BestIndividualSelection ♦	*SelectionOperator* ♦

[1]Italics used to indicate abstract classes
♦JGA built-in class
•User-defined class

The configuration file

JGA has a main configuration file that is used to set up the application parameters. This file is responsible for all of JGA settings and it also points to the files that contain the specific information of the problem instance that we want to solve.

The configuration file used for the Ackley problem, called `JGAConfigAckley.ini` is shown in Figure 16. This file is located at `[target]\examples\Ackley`.

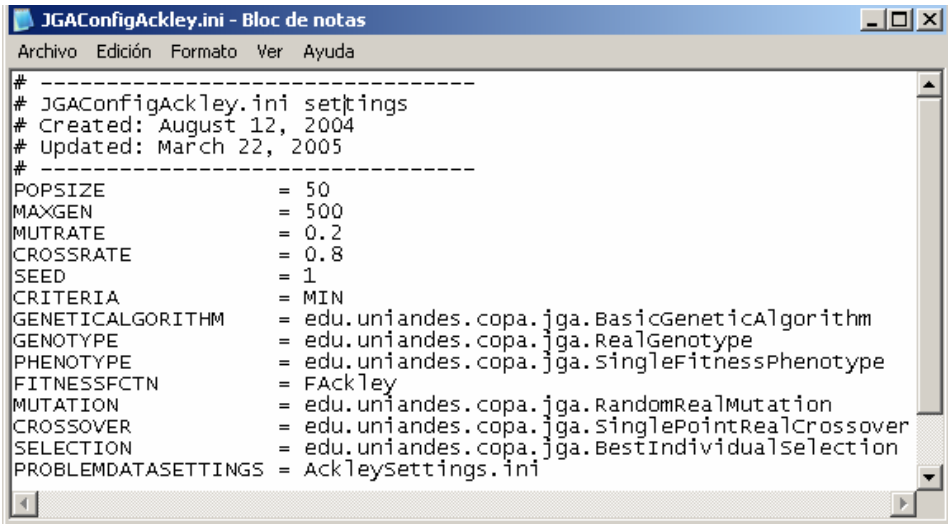

Figure 16: Main configuration file for the Ackley Problem (JGAConfigAckley.ini)

The configuration file is a text file with a field-value structure. Table 3 describes the fields in the configuration file.

Table 3: Fields in the main configuration file

Property Field	Description
POPSIZE	Sets the population size. Its value is an integer greater than or equal to 1.
MAXGEN	Sets the maximum number of generations. Its value is an integer greater than or equal to 1.
MUTRATE	Sets the probability of mutation. Its use depends on the implementation of the mutation operator or the logic of the algorithm (extension of *GeneticAlgorithm*). It could be the probability that a given gene mutates or the probability that a whole individual of the population mutates. Its value is a real number between 0 and 1.
CROSSRATE	Sets the probability of crossover. Its use depends on the logic of the algorithm (extension of *GeneticAlgorithm*). For instance, in BasicGeneticAlgorithm, it is the probability that a given individual of the population is chosen as a parent, forming the so-called crossover pool. Its value is a real number between 0 and 1.
SEED	Sets the seed for the random number generator. It characterizes the execution of an independent run of the genetic algorithm. Its value is a non-negative integer.
CRITERIA	Sets the optimization criteria for the objective. Its value may be either MIN or MAX, for minimization or maximization, respectively.
SELECTION	Sets the class name with the implementation of the selection mechanism.
PROBLEMDATASETTINGS	Sets the file name with the problem-specific configuration file.
GENETICALGORITHM	Sets the class name with the main logic of the genetic algorithm.
GENOTYPE	Sets the class name that defines how a solution is coded into a genotype. It must extend from class Genotype.
PHENOTYPE	Sets the class name that defines and stores the individual's phenotype. It must extend from class Phenotype.
FITNESSFCTN	Sets the class name with the implementation of the fitness function evaluation.

	Usually this is a problem-specific function that knows how to evaluate a specific genotype.
MUTATION	Sets the class name with the implementation of the mutation operator.
CROSSOVER	Sets the class name with the implementation of the mutation operator.
SELECTION	Sets the class name with the implementation of the selection mechanism.
PROBLEMDATASETTINGS	Sets the file name with the problem-specific configuration file.
OUTPUTLEVEL	Sets the verbosity level for output messages. Depending on the type of messages to be displayed, Its value is the result of the sum of the following values: 2^0=1, for trace messages; 2^1=2, for warning messages; 2^2=4, for debug messages; and 0, for silent output. For example, a property value of 5 (=1+4) turns on trace and debug messages.
STATISTICSCOLLECTOR	Sets the type of statistics to be collected during the genetic algorithm execution- Its value is the result of the sum of the following values: 2^0=1, for Collecting basic statistics such as the number of function evaluations, mutation, and crossover operations. 2^1=2, for tracking the best individual during the execution of the genetic algorithm. It also collects the generation number and the number of fitness function evaluations when the best individual was found. 2^2=4, for Recording the best and worst individuals for each generation. For example, a property value of 3 (=1+2) turns on basic and best individual statistics.

For the Ackley problem, the problem settings file (named AckleySettings.ini) contains the number of variables (dimension of the solution vector) and the bounds for each variable (see Figure 17).

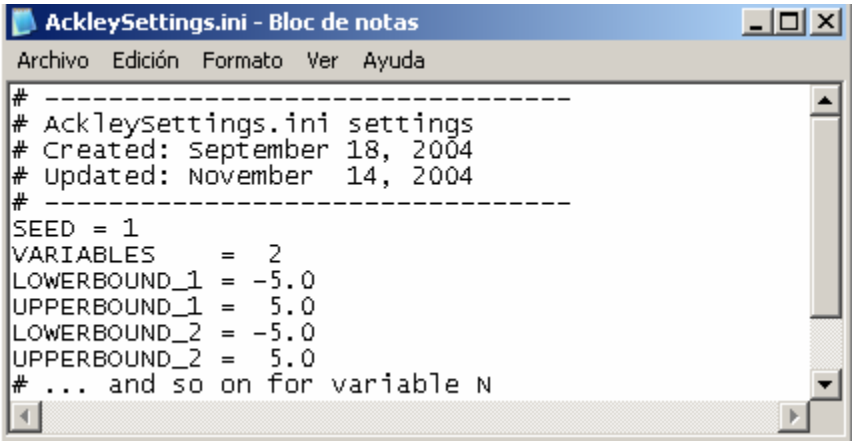

Figure 17: Problem settings file for Ackley Problem

REFERENCES

D. H. Ackley D.H.. "A connectionist machine for genetic hillclimbing". Boston: Kluwer Academic Publishers, 1987.

Gen M., Cheng R., Genetic Algorithms and Engineering Optimization, John Wiley and Sons, 2000

Holland J.H., Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.

Pham D.T., Karaboga D., Intelligent Optimization Techniques, Springer, 2000.

Rayward-Smith V.J., Soma I.H.. Reeves C.R., Smith G.D, Modern Heuristic Search Methods, John Wiley and Sons, 1996.

Michalewicz Z., Genetic algorithms + data structures = evolution programs, 3rd ed., Springer, 1996.

CONTACT INFORMATION

For further technical information or to report any problem related to JGA, please contact:

Andrés L. Medaglia, Ph.D.
amedagli@uniandes.edu.co
JGA designer & developer

Eliécer Gutiérrez G. M.Sc.
egutierr@uniandes.edu.co
JGA developer