

jGE Library Quick Start Guide

Technical Requirements

In order to use jGE v1.0, you will need Java 5 or later.

Examples

In the source code, you will find a lot of usage examples. Alongside with the jGE API documentation it will help you to start using jGE. Both of them are freely available from the jGE web site (aiia.bangor.ac.uk/jge).

Namely, the source file `bangor/aiia/jge/core/EAExperiments.java` contains some examples of using the jGE classes. Indeed, the source folder `bangor/aiia/jge/junit` contains all the JUnit (JUnit.org, 2006) tests of jGE. These JUnit TestCases can serve as a rich resource of how to use the various classes of the library.

Just keep in mind that anywhere in the source code you see methods like “`ConfigurationSettings.getInstance().getxxx`”, replace them with a String containing the actual path of the corresponding file or folder in your system. For your convenience, it is suggested that you change the settings of the class `bangor.aiia.jge.util.ConfigurationSettings` to the default values of your system and recompile the class.

Sample Code using Standard GA in a Hamming Distance Problem

```
// The binary string we are looking to find (the solution)
String target = "1110001110001010101010101010";

// The Problem Specification (implementation of Evaluator)
HammingDistance hd = new HammingDistance(target);

// Instantiation of the Evolutionary Algorithm (SGA)
StandardGA ga = new StandardGA(50, 1, 30, 30, hd);

// Configure the Standard Genetic Algorithm
ga.setFixedSizeGenome(true);
ga.setCrossoverRate(0.9);
ga.setMutationRate(0.01);
ga.setDuplicationRate(0.01);
ga.setPruningRate(0.01);
ga.setMaxGenerations(100);
```

```
// Execute the Evolutionary Algorithm (SGA)
// and get the solution
Individual<String, String> solution = ga.run();
```

Sample Code using Grammatical Evolution in a Hamming Distance Problem

```
// The binary string we are looking to find (the solution)
String target = "1110001110001010101010101010";

// The Problem Specification (implementation of Evaluator)
HammingDistance hd = new HammingDistance(target);

// Instantiate the BNF Grammar (use for this example
// the file BinaryGrammarFixedLength.bnf)
BNFGrammar bnf = null;
try {
    bnf = new BNFGrammar(
        new File("[THE_PATH_OF_THE_BNF_GRAMMAR_FILE]")
    );
}
catch(IOException ioe) {
    System.out.println("IOException: " + ioe.getMessage());
}

// Instantiation of the Evolutionary Algorithm (SGA)
GrammaticalEvolution ge = new GrammaticalEvolution(
    bnf, hd, 50, 8, 20, 40
);

// Configure the Grammatical Evolution Algorithm
ge.setCrossoverRate(0.9);
ge.setMutationRate(0.01);
ge.setDuplicationRate(0.01);
ge.setPruningRate(0.01);
ge.setGenerationGap(0.9);
ge.setMaxGenerations(100);

// Execute the Evolutionary Algorithm (GE) and get the solution
Individual<String, String> solution = ge.run();
```

Evaluator

In order to use jGE for ad-hoc problems, you have just to implement a class which will evaluate the individuals of the population (assign a fitness value). This evaluator must implement the `bangor.aiia.jge.core.Evaluator` interface and actually will “encapsulate” the Problem Specification.

Such “out of the box” classes (available in jGE) are the following:

- `bangor.aiia.jge.ps.HammingDistance`

- `bangor.aiia.jge.ps.SymbolicRegression`

For other classes of problems, just create your own implementation of the Evaluator interface and use it with the evolutionary algorithm of your choice (Standard GA, Steady-State GA, or GE).

Mapper

Regarding the mapping, in case of Standard Genetic Algorithms there is no distinction between genotype and phenotype. Consequently, if you do not set a specific implementation of `bangor.aiia.jge.core.Mapper` interface (you can create your own) the default “no-mapping” implementation will be used (`bangor.aiia.jge.core.DefaultMapper`). Then, you have to “decode” the semantics of the binary string of the genotype in your implementation of the Evaluator.

Grammatical Evolution

In order to use Grammatical Evolution with jGE in your project, you have to create an instance of the `bangor.aiia.jge.core.GrammaticalEvolution` class and pass to it both the file with the BNF Grammar you will use and your own implementation of the `bangor.aiia.jge.core.Evaluator` interface (which will evaluate the fitness of an Individual of the Population).

Namely, you have to implement a BNF Grammar (which dictates the valid phenotypes of the individuals) and the Evaluator (which assigns a fitness value to each individual).

In the accompanying CD, you will find some sample BNF Grammars. You can use them to play around with jGE until you become more familiar and start to create your own.