Analog Circuit Design using Grammatical Evolution (ACID-GE)

# Introduction

This document describes the Analog Circuit Design using Grammatical Evolution software (ACID-GE). It covers some source code description, as well as distribution description, including instructions for its use.

A former version of the ACID-GE software was developed and used for all the circuits synthesized in the following paper, published in Applied Soft Computing:

Federico Castejón, Enrique J. Carmona,

Automatic design of analog electronic circuits using grammatical evolution,

Applied Soft Computing,

Volume 62, 2018, Pages 1003-1018, ISSN 1568-4946,

https://doi.org/10.1016/j.asoc.2017.09.036.

# Dependencies

ACID-GE software has been developed and tested in Ubuntu Linux, using the following tools:

* Maven 3.3.9
* Eclipse Oxygen
* Java: Oracle JDK 8u171
* NGSpice revision 27
* javaCC v5.0

# Source code

The source code uses Maven as the tool for compiling and building the library. The source code is organized in a standard directory structure, which is described as follows:

acidge

│

├── src

│   ├── main

│   │   ├── java <--- Java source code

│   │   ├── jjtree <--- sources for jjtree (Javacc)

│   │   ├── resources <--- scripts, grammar files and properties

│   │   └── assembly <--- dist.xml for building distribution file

│ │

│   └── test

│      ├── java <--- Junit test cases

│      └── resources <--- grammar files and properties for testing

│

├── target <--- Binaries are built here

│

├── pom.xml <--- Maven project object model file

├── NOTICE.txt <--- Notice.txt file

├── README.txt <--- Readme.txt file

└── LICENSE.txt <--- License information

# Package description

This section covers a shallow description of the package structure of ACID-GE. The base package for all the source code is:

es.uned.simda.acidge

## es.uned.simda.acidge.ge

This package comprises the main classes for the algorithm, some of them are the following:

* Main: starting class
* GE: implements the evolutionary algorithm engine
* Poblacion: population class comprising Fenotipo instances
* Fenotipo: Fenotype
* Genotipo: genotype or chromosome class

## es.uned.simda.acidge.ge.cache

This package implements a cache, to avoid evaluating a chromosome twice. The key is a hash value of the generated netlist.

This cache also implements a control for parallel execution.

## es.uned.simda.acidge.ge.operadores

This package comprise all operators used by the algorithm, including mutation, crossover, initialization, parent selection and survivor selection operators.

## es.uned.simda.acidge.ge.random

This class implements a random number generation interface for wrapping different implementing classes.

## es.uned.simda.acidge.ge.random.mersenne

This package includes MersenneTwisterFast class which implements Mersenne twister random number generation. Refer to that file for the authors and license.

## es.uned.simda.acidge.generador

This package implements a grammatical evolution decoder. It decodes a byte array from the chromosome and generates a final expression. The main decoder class is GeneradorGrammar.

## es.uned.simda.acidge.generador.gramatica

This package implements an abstract grammar which is used to decode chromosomes by the grammatical evolution algorithm. The EBNF grammar parser reads a file and generates a Gramatica Class from this package.

## es.uned.simda.acidge.generador.parser

This package implements an EBNF grammar parser, which is used for grammar file reading. The parser is based on Javacc and JJtree. It allows to write an abstract syntax tree (AST) from the grammar file read.

## es.uned.simda.acidge.generador.arbol

This package implements a tree structure to be used while decoding a chromosome.

## es.uned.simda.acidge.netlist

A raw decoded netlist cannot be fed to NGSpice as it is. A post-processing is needed to get the final netlist. This package implements this post-processing.

## es.uned.simda.acidge.problema

This package implements an interface for evaluating chromosomes to solve a specific problem.

* Problema: an interface to wrap different kind of problems to be solved by the algorithm
* EvalRes: a superclass to contain a chromosome evaluation result

## es.uned.simda.acidge.problema.dev

This package implements a circuit constructor class and a subclass a Problema for circuit developing.

## es.uned.simda.acidge.problema.dev.eval

This package implements a superclass DevEval to evaluate different kind of circuits. KozaEvalRes is a subclass of EvalRes, and is used to contain a circuit evaluation result.

## es.uned.simda.acidge.problema.dev.eval.cc

This package implements specific classes for evaluating computational circuits. A specific class is provided for squaring, square root, cubing and cube root circuits.

## es.uned.simda.acidge.problema.dev.kozamat

This package implements specific classes for evaluating non-computational circuits. A specific class is provided for temperature sensing circuit, voltage reference circuit and gaussian function circuit.

## es.uned.simda.acidge.problema.dev.netlist

This class implements a subclass of circuit constructor for generating netlists.

## es.uned.simda.acidge.rmi

Classes for parallel execution, which is based on RMI.

## es.uned.simda.acidge.spice

Classes for NGSpice calling and retrieving results. Main classes are:

* Spice: multithreading class for calling an NGSpice process and reading its output.
* ProcesaSalidaSpice: process NGSpice output and generates HashMap of Signal class
* Signal: it is a vector of values read from NGSpice output for processing in evaluation classes

## es.uned.simda.acidge.stats

Utility class for statistics generation.

## es.uned.simda.acidge.util

Logging classes.

# Distribution

This section describes the directory structure for the distribution package. This directory allows algorithm run and testing.

A compressed tar file (.tgz) is built with the jar file, and all scripts, grammar files and properties files. The name of this file is: acidge-1.0.0-SNAPSHOT-distribution.tgz. Uncompressing of this file produces the following directory structure:

acidge-dist

│

├── bin <--- scripts for executing the algorithm

│

├── lib

│   └── acidge.jar <--- library comprising the algorithm binaries

│

├── grammar <--- grammar files in EBNF format

│   └── various ebnf files

│

├── properties

│   ├── sensor <--- properties for temperature sensing circuit

│   │   ├── GE.properties

│   │   └── netlist.properties

│   │

│   ├── vref <--- properties for voltage reference circuit

│   ├── gaussian <--- properties for gaussian function circuit

│   ├── square <--- properties for squaring circuit

│   ├── sqroot <--- properties for square root circuit

│   ├── cube <--- properties for cubing circuit

│   └── cuberoot <--- properties for cube root circuit

│

├── logs <--- log files

│

└── 20180504\_... <--- temporal directories

# Properties files

This section describes the properties files used by the algorithm. These files are GE.properties y netlist.properties. The algorithm looks for these files in the properties directory. Several example properties files are provided. The set of files to be used, need to be copied to the properties directory before running the algorithm.

## GE.properties

This file comprises the properties for the algorithm. The properties are as follows:

* Paralelismo: execution in parallel. true or false.
* GeneratorClassName: fully qualified class name of Generator. It is already set to the appropriate class and should not be changed.
* ProblemClassName: fully qualified class name of problem to solve implementing class. It is already set to the appropriate class and should not be changed.
* CircuitConstructorClassName: fully qualified class name of circuit constructor class It is already set to the appropriate class and should not be changed.
* DevEvalClassName: fully qualified class name of circuit evaluation class. There is one class for each circuit tested. See the example properties files.
* IndividualsNumber: Number of chromosomes in the population. In the circuits tested it is set to 1000.
* MinGenesNumber: minimum chromosome length in random initialization.
* MaxGenesNumber: maximum chromosome length in random initialization.
* LimitMaxGenesNumber: maximum chromosome length. It is used in crossover operators.
* MaxEvaluationCount: alternative algorithm termination condition. Algorithm ends when this evaluation count is reached. It is not used in the circuits tested.
* GenerationsNumber: algorithm ends when this generation number is reached.
* TerminationConditionType: it is set to 5 in the circuits tested, which marks the first successful chromosome, and allows the algorithm to reach the maximum generations.
* PopulationInitializationClassName: fully qualified class name of population initialization operator class. It is already set to the appropriate class and should not be changed.
* ParentSelectionClassname: fully qualified class name of parent selection operator class. It is set to tournament selection for the circuits tested.
* TournamentSize: tournament size. It is set to 3 for the circuits tested.
* CrossoverClassName: fully qualified class name of crossover operator class. It is set to one-block operator for the circuits tested.
* CrossoverRate: crossover rate. It is set to 0.5 in the circuits tested.
* CrossoverPointsNumber: crossover points number. It should be set to one for the selected operator.
* CrossoverBlockSize: crossover block size. It has to agree with the block size of the grammar used.
* CrossoverRateGE2: not used
* CrossoverRateGE3: not used
* MutationClassName: fully qualified class name of mutation operator class. It is set to bitwise mutation for the circuits tested.
* MutationRate: mutation rate. It is set to 0.001 for the circuits tested
* MurationRateGE2: not used
* DuplicationClassName: fully qualified class name of duplication operator class. It is set to the appropriate class and should not be changed.
* DuplicationRate: duplication operator rate. It is set to zero for the circuits tested, since it is not really used.
* SurvivorSelectionClassName: fully qualified class name of survivor selection operator class. It is set to generational selection for the circuits tested.
* Elitism: number of the best chromosomes to save from one generations to the next. It is set to two for the circuits tested.
* GenerationalGap: number of chromosomes to replace on each generation, in steady-state survivor selection. It is not used for the circuits tested.
* MaxWrappingNumber: maximum wrapping allowed. It is set to 4 for the circuits tested.
* GrammarFileName: grammar filename. It should be in the grammar directory.
* FunAdaptacionClassName: fully qualified class name for fitness mapping class. It is not used for the circuits tested and should not be changed.
* UmbralError: not used
* GoalFitness: fitness goal objective. Not used in circuits generation which use its own criterium coded in evaluation class.
* HasGoal: wether problem has a goal. True or false. It is not used in circuits generation, and should be set to false.
* RandomGeneratorClassName: fully qualified class name for random number generation class. It is set for Mersenne twister random generation for the circuits tested.
* MaxRecursionLevel: maximum recursion level in grammar decoding. It should not be changed.
* HashCache: Fitness cache using. It is set to true for the circuits tested.

## netlist.properties

This file comprises the properties for netlist building. The properties are as follows:

* Simulaciones: number of simulations. It is set to one for the circuits tested.
* NodosProtegidos: a set of nodes separated by commas. If one of these nodes is left dangling or unconnected, the circuit will be considered unfeasible.
* EvitarComponentesColgando: if this property is set to one, a high valued resistor is connected from every dangling node to ground.
* ResistenciaElevada: this property is the value for the high valued resistor to be connected from dangling node s to ground. It is set to 1G for the circuits tested.
* netlist.header.0: this property contains the header lines to be appended at the top of the generated netlist. This parts contains the fixed part of the circuit. It is a multiline property.
* netlist.modelo: this property contains the transistor models used. It is a multiline property.
* netlist.analisis.0: this property contains the NGSpice commands setting the analysis to be done in the simulation. It is a multiline property.

# server.properties

This is an additional properties file used for parallel execution. This file contains the relation of servers in a cluster of nodes for parallel execution. It contains one line per node, indicating the name of the node and the number of server processes started on it, separated by a ‘=’ character, as the following example:

node1=4

node2=3

# Post-processing

This section describes the post-process done to a raw netlist generated by the decoding of the chromosome to prepare it for NGSpice simulation.

1. Inserting netlist.header.0 at the beginning of the netlist. This contains the fixed part of the circuit.
2. Appending netlist.modelo which contains transistor models used.
3. Appending netlist.analisis.0 which contains NGSPice commands
4. Deleting strings null1 and null2
5. Checking for protected nodes not to be dangling
6. Connecting 1GΩ resistors from dangling nodes to ground
7. Removing short circuited components
8. Converting component values from scientific notation to standard notation with suffixes.
9. Renumbering components

# Scripts

This section describes several scripts which are used to run the algorithm. They are all in the bin directory.

* ejecutar.sh: basic script to launch an algorithm run.
* lanza.sh: script for launching several runs. It takes a number parameter which stands for the number of runs.
* nhlanza.sh: script for launching several runs as the former script. This script uses nohup command to allow disconnecting from the SSH session without interrupting the execution.
* filtra.sh: a script to extract useful information from the log files generated in a set of runs.
* server.sh: used to start a server used in parallel execution.
* nhserver.sh: used to start a number of servers with nohup, in parallel execution.

# Parameters

Some parameters can be passed to the algorithm. They are the following:

* -server: it starts the program as a server for parallel execution
* -s number: used in conjunction with -server flag. It sets the server number
* -c value: it sets the crossover rate
* -m value: it sets the mutation rate

# Log files

This section describes the log files generated by the algorithm during a run.

* salida\_yyyymmdd\_hhmiss\_xxx.log: log file from an algorithm run.
* salserver\_m\_n: log file from server m. It is rotated so n can be 0 or 1.

While the algorithm is running, the following command can be used to monitor generations and best fitness so far:

* tail -f salida\_yyyymmdd\_hhmiss\_xxx.log | grep BestFitness

The contents of the GE.properties file used are dumped to the log file when the algorithm start.

When the algorithm ends, the best fitness is reported in the log file with a line which start with the string “Mejor”, followed by the fitness value, number of hits obtained. After that, the raw netlist generated is included, which comprise several lines. At the end of the netlist, some information of the chromosome is included, comprising the generation in which the chromosome appeared, the expressed length, and the complete byte string.

This raw netlist cannot be directly used as an input for the NGSpice simulator. The final netlist is reported in the log file with a line which start with the string “BestCircuit”. The following lines comprises the netlist which can be directly used in the simulator.