# AGE Documentation: addendum for version 1.1

Adam Nohejl

24 November 2011

# Chapter 1

# Introduction

This addendum documents new features, including new applications, in AGE version 1.1. The remaining documentation is in the `Documentation.pdf` file based on my bachelor thesis. I hope I will get round to putting it all together and separating from the general stuff and experiments in my bachelor thesis (there is currently a lot of cross-references). I also want to add relevant information from my master thesis (Nohejl, 2011).

# Chapter 2

# User Documentation: Additions in version 1.1

## 2.1 Command line interface

Square brackets denote optional command line options. The following options have been added in version 1.1:

### Algorithm options

[`-T`] forces generation of derivation trees (normally not generated for GE) necessary to compute bushiness statistics.

### Execution options

[`-F`] halts a sequence of runs when success (as specified by **`-f`**) is reached.

### File output options

[`-i`] includes only the best-of-generation individual the XML output.

## 2.2 Implemented components

Each component is listed under the name of the class that implements it, which is followed by a description and three short sections:

Argument: The argument form to be used for the corresponding command line option in order to specify an algorithm element. Arguments that do not have any options are simple labels, such as **`roulette`** for roulette-wheel selection. A roulette-wheel selection is therefore specified by `-S roulette`. Arguments that have parameters are in one of the following forms:

$$\textit{label}(m_1, \ldots, m_M)$$

$$\textit{label}(m_1, \ldots, m_M[, o_1, \ldots, o_N])$$

$$\textit{label}([o_1, \ldots, o_N])$$

where *label* identifies the algorithm element, $m_1$ to $m_M$ are mandatory parameters, and $o_1$ to $o_N$ are optional parameters, which have default values. Parentheses and commas are part of the syntax. The parameters are positional, which means that you can supply the first $n \leq N$ of the optional parameters, but not for instance only the first one and the third one. If there are no mandatory parameters and you do not supply any optional ones, the enclosing parenthesis can be omitted. If you invoke AGE from a standard Unix shell, any parentheses will have to be quoted or escaped.

For instance the argument **`tour`**`(`[*t*]`)` for tournament selection means that the tournament selection has one optional parameter *t* (the tournament size). Tournament selection can therefore be specified as `-S tour` to use the default tournament size, or as `-S tour(4)` to use a size of 4, in which case the command line options will usually have to be entered as `-S 'tour(4)'` or `-S tour\(4\)`.

Interface: The source file that contains the interface of the component. Recursively included header files are not listed. Additionally, all implemented algorithm elements can be including using the convenience header file `Elements.h`.

Implementation: The source files that contain the implementation of a component. The files the implementation depends on are listed only if they are not part of the AGE library.

This following components have been newly implemented or updated.

### 2.2.1 Initialisers

**RampedInitialiser** implements a generalisation of the "sensible" initialisation method, which in turn is based on Koza's ramped half-and-half initialisation. See `Documentation.pdf` for more information. (Only the *cd* option has been added in v1.1.)

Argument: **`ramped`**`(`*m*–*n*[`,`*stoch*`,`*grow*`,`*oneill*`,`*u*`,`*tl*`,`*tr*`,`*cd*]`)`

*m* to *n* (inclusive) is the range of derivation tree heights, $m = 0$ is replaced by the lowest height possible for a given grammar;

*stoch* indicates whether the "grow" rate is to be interpreted as a probability (`1`), or a ratio (`0`, default);

*grow* is the "grow" rate from $[0, 1]$, default: 0.5;

*oneill* indicates whether recursive productions receive special treatment "O'Neill-style" (`1`), or not (`0`, default), see **??**;

*u* indicates whether each generated tree is unique (`1`), or not (`0`, default);

*tl* is the absolute length of the tail of random codons (default: 0);

*tr* is the ratio of the tail length to the significant part length (default: 0).

*cd* **(new in v1.1)** indicates whether codon-level degeneracy is applied (default: 1).

At least one of *tl* and *tr* must be 0.

Interface: `RampedInitialiser.h`

Implementation: `RampedInitialiser.cpp`

**CFGGPInitialiser** implements the "grow" initialisation method for CFG-GP (Whigham, 1995). Also see Nohejl (2011).

Argument: **`cfggp`**`(`*maxHeight*`[,`*u*`])`

*maxHeight* is the maximum height of generated derivation trees;

*u* indicates whether each generated tree is unique (`1`), or not (`0`, default);

Interface: `CFGGPInitialiser.h`

Implementation: `CFGGPInitialiser.cpp`

### 2.2.2 Fitness scalings

**GreedyFitnessScaling** implements the greedy overselection technique used by Koza (1992) for large populations as a fitness scaling. Automatically uses the quantitative parameters used by Koza according to population size. It should be used in conjunction with roulette-wheel selection. Results of Koza's original implementation may depend on the sort algorithm if some individuals with an "edge" fitness value do not fit. The just parameter can be used to include them at the cost of somewhat diluting the overselection.

Argument: **`greedy`**`([`*just*`])`

*just* indicates whether to force the same overselection of individuals with equal fitness values (`0`, default).

Interface: `FitnessScalings.h`

Implementation: `FitnessScalings.cpp`

### 2.2.3 Crossover operators

A probability of any crossover operator (incidentally only one is currently implemented) is set by the **`-x`** command line option, separately from other parameters.

**CFGGPCrossover** implements CFG-GP tree-based crossover (Whigham, 1995). Also see Nohejl (2011).

Argument: **`cfggp`**`([`*ir*`,`*alt*`])`

*ir* is the probability from $[0, 1]$ of picking an internal node for crossover, or alternatively $-1.0$ to pick nodes regardless of type, default: -1;.

*alt* indicates the type of node alternative to internal: 1 means external, 0 means any, default: 1.

Interface: `CFGGPOperators.h`

Implementation: `CFGGPOperators.cpp`

### 2.2.4 Mutation operators

A probability of mutation operators is set by the **`-m`** command line option, separately from other parameters.

**CFGGPMutation** implements CFG-GP tree-based mutation (Whigham, 1995). Also see Nohejl (2011).

Argument: **`cfggp`**`(`[*ir*, *alt*, *maxHeight*]`)`

- *ir* is the probability from $[0,1]$ of picking an internal node for crossover, or alternatively $-1.0$ to pick nodes regardless of type, default: -1;.
- *alt* indicates the type of node alternative to internal: 1 means external, 0 means any, default: 1.
- *maxHeight* is the maximum height of newly generated subtrees, 0 denotes the height of the subtree that is being replaced, default: no limit.

Interface: `CFGGPOperators.h`

Implementation: `CFGGPOperators.cpp`

### 2.2.5 Applications

**DotProductFitnessEvaluator** implements dot product symbolic regression (Wong and Leung, 2000), used for experiments in my master thesis (Nohejl, 2011).

Argument: `dot(`[*variant*, *maxWraps*, *maxHeight*, *bnf*]`)`

- *variant* identifies the variant of the problem, either `dot` (default, the dot product) or `adf` (the expression $\vec{x} \cdot \vec{y} + \vec{y} \cdot \vec{z}$).
- *maxWraps* is the maximum number of wrapping events, default: 3.
- *maxHeight* is the maximum tree height, default: 20.
- *bnf* is path to the BNF grammar file, default: based on variant.

Interface: `App-DotProductFitnessEvaluator.h`

Implementation: `App-DotProductFitnessEvaluator.cpp`

**BooleanParityFitnessEvaluator** implements the boolean parity symbolic regression (Koza, 1992), used for experiments in my master thesis (Nohejl, 2011).

Argument: `parity(`[*odd*, *arity*, *maxWraps*, *maxHeight*, *bnf*]`)`

- *odd* indicates whether to target the odd (1) or even (0, default) parity function.
- *arity* is the arity of the parity function, default: 3.
- *maxWraps* is the maximum number of wrapping events, default: 3.
- *maxHeight* is the maximum tree height, default: 20.
- *bnf* is path to the BNF grammar file, default: a simple GP-like grammar without ADFs based on arity.

Interface: `App-BooleanParityFitnessEvaluator.h`

Implementation: `App-BooleanParityFitnessEvaluator.cpp`

**TimeTablingFitnessEvaluator** implements the exam timetabling problem (Bader-El-Den et al., 2009), used for experiments in my master thesis (Nohejl, 2011).

Argument: `tt(`[*alpha*`,` *ttData*`,` *maxWraps*`,` *maxHeight*`,` *bnf*]`)`

*alpha* is the $\alpha$ constant for the fitness formula (Bader-El-Den et al., 2009), default: 100000.

*ttData* is the problem data file, it can be any of `car91`, `car92`, `ear83`, `hec92`, `kfu93`, `lse91`, `sta83`, `tre92`, `uta92`, `yor83` (Carter et al., 1996) or an external path, default: `sta83`.

*maxWraps* is the maximum number of wrapping events, default: 3.

*maxHeight* is the maximum tree height, default: 20.

*bnf* is path to the BNF grammar file, default: based on (Bader-El-Den et al., 2009).

Interface: `App-TimeTablingFitnessEvaluator.h`

Implementation: `App-TimeTablingFitnessEvaluator.cpp`, `App-TimeTabling.h`, `App-TimeTabling.cpp`

# Bibliography

Mohamed Bahy Bader-El-Den, Riccardo Poli, and Shaheen Fatima. Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing*, 1(3):205–219, 2009.

Michael W. Carter, Gilbert Laporte, and Sau Yan Lee. Examination timetabling: Algorithmic strategies and applications. *J Oper Res Soc*, 47(3):373–383, 03 1996. URL `http://dx.doi.org/10.1057/jors.1996.37`.

John R. Koza. *Genetic programming: On the programming of computers by natural selection*. MIT Press, 1992. ISBN 0-262-11170-5.

Adam Nohejl. *Grammar-based genetic programming*. Master thesis, available at `http://nohejl.name/age/pdf/Adam-Nohejl-2011-Grammar-based-GP.pdf`, 2011.

Peter Whigham. Inductive bias and genetic programming. In *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, pages 461–466, 1995.

Man Leung Wong and Kwong Sak Leung. *Data Mining Using Grammar-Based Genetic Programming and Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 2000. ISBN 079237746X.