PgafFrame — User Documentation

Jordi Petit

Report LSI-97-9-T
PgaFrame
User Documentation
(Draft)

Jordi Petit i Silvestre
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
http://www-lsi.upc.es/~jpetit/PgaFrame

Overview

This page is the user documentation of PgaFrame - a frame for parallel genetic algorithms. It is made of two parts: the first describes the abstract parameters of the PgaFrame; the second describes the functions that users can use to write their functions. Another purpose of this page is to show by the example how to document an abstract frame using the Frames approach.

This document has been written to be read on-line. It can be found at http://www-lsi.upc.es/~jpetit/PgaFrame/PGA.html on the WWW.

Abstract parameters

In this section the meaning of the abstract parameters of the PgaFrame is described. Please, see the abstract file PGA.abs.

General parameters

The out types

```c
typedef struct PGAClass {
    // PgaFrame module
    PgaFrame *pframe;
    // PgaContext module
    PGAClass *pcontext;
    // PgaContextPtr module
    PgaContextPtr pcontext_ptr;
} PGAClass;
```

The PGAClass, PgaContext, and PgaContextPtr types are the abstract data types that the frame provides.
Type of alleles

TYPE DataTypeEnum = enum (Binary, Integer, Real, Character);
CONSTANT DataTypeEnum DataType;

This required parameter is needed to specify the type of the alleles that make up a chromosome. The possible values are:

- Binary: binary-valued strings
- Integer: integer-valued strings
- Real: real-valued strings
- Character: character-valued strings

In the future, a new value for user-defined strings will be available.

Direction of the optimization process:

TYPE OptDirectionEnum = enum (Maximize, Minimize);
CONSTANT OptDirectionEnum Direction;

This required parameter is needed to specify the direction in which the objective function should be optimized: minimization or maximization. The possible values are:

- Maximize: to indicate a maximization problem
- Minimize: to indicate a minimization problem

Evaluation function

FUNCTION double Evaluate (PGACtxPtr ctx, int p, int pop);

This required parameter must be supplied by the user to compute the evaluation of an individual chromosome. The value returned by this function must be a double value corresponding to its evaluation.

Preprocess function

FUNCTION int Preprocess (int argc, charPtrPtr argv);

This required parameter must be supplied by the user to specify the length of the chromosomes. It can also be used to initialize the genetic algorithm, for example by reading its input. In any case, the return of this function must be an int specifying the chromosome length.

Postprocess procedure

OPTIONAL PROCEDURE Postprocess (int argc, charPtrPtr argv, PGACtxPtr ctx);

This parameter can be supplied by the user in order to enable the genetic algorithm to do a postprocessing of its result. For example, it can be used to save the best values found.

UserCode statement

OPTIONAL STATEMENT UserCode ();

This statement will be included at the beginning of the code, in the global scope. It can be given by the user in order to declare data types, global variables, global functions, etc. that (s)he will use in the rest of functions.
Population replacement

Population size

```c
OPTIONAL CONSTANT int PopulationSize = "100";
```

Specifies the size of the genetic algorithm population. The default population size is 100.

Replacement value

```c
OPTIONAL CONSTANT int ReplacementValue;
```

Specifies the number of new strings to create at each generation. The default is ten percent of the population size.

Replacement type

```c
TYPE ReplacementTypeEnum = enum {ReplaceBest, ReplaceRandom, ReplaceRandomNoRep};
OPTIONAL CONSTANT ReplacementTypeEnum ReplacementType = "ReplaceBest";
```

This parameter selects the method of sorting strings to copy from the old population to the new population. Valid choices are:

- **ReplaceBest**: for copying the best strings
- **ReplaceRandom**: for copying random strings with replacement
- **ReplaceRandomNoRep**: for copying random strings without replacement

Allowing duplicates in population

```c
OPTIONAL CONSTANT Boolean AllowDuplicates = "False";
```

A boolean flag to indicate if duplicate strings are allowed in the population.

Checking of duplicate strings

```c
OPTIONAL FUNCTION int DuplicateChecking (PGAContextPtr ctx, int pi, int pop);
```

This function can be written by the user in order to override the default behaviour of checking if two strings are equal in a population.

Stopping criteria

UserDefinedStopRule

```c
OPTIONAL CONSTANT Boolean UserDefinedStopRule = "False";
```

Specifies if the stopping rule is given by the user or, in the contrary is one of the predefined ones.

StopMaxIter / MaxIterValue

```c
OPTIONAL CONSTANT Boolean StopMaxIter = "True";
OPTIONAL CONSTANT int MaxIterValue = "1000";
```
Stop when maximum number of iterations is reached. The maximum number of iterations is given by the MaxIterValue parameter. This rule can be combined with the other predefined stopping rules.

**StopNoChange / MaxNoChangeValue**

```c
OPTIONAL CONSTANT Boolean StopNoChange = "False";
OPTIONAL CONSTANT int MaxNoChangeValue = "100";
```

Specifies that the GA will stop when during a certain number of iterations there is no change in the best evaluation. The number of no change iterations is given by the MaxNoChangeValue parameter. This rule can be combined with the other predefined stopping rules.

**StopTooSimilar / MaxSimilarValue**

```c
OPTIONAL CONSTANT Boolean StopTooSimilar = "False";
OPTIONAL CONSTANT int MaxSimilarValue = "95";
```

Specifies that the GA will stop when the percentage of population with same evaluation function similarity is close. The similarity percentage is given by the MaxSimilarValue parameter. This rule can be combined with the other predefined stopping rules.

**StopCondition function**

```c
OPTIONAL FUNCTION int StopCondition (PGAContextPtr ctx);
```

When `userDefinedStopRule` is set to `True`, this function must be written in order to decide the termination of the algorithm. This rule cannot be combined with the predefined stopping rules.

**Initialization**

**Initialization type**

```c
TYPE InitializationTypeEnum = enum (InitZero, InitBinRandom, InitCharLower, 
OPTIONAL CONSTANT InitializationTypeEnum InitializationType;
```

Strings are initialized randomly (the default), or set to zero. The choice is specified by setting this parameter. The possible values and their meaning are:

- **InitZero**: Each gene will be set to zero.
- **InitBinRandom**: Each gene will be set to zero or one, according to the probability specified by the `BinInitProbBit1` parameter.
- **InitCharLower**: Each gene will be set to a lowercase alphabetic character chosen uniformly random.
- **InitCharUpper**: Each gene will be set to an uppercase alphabetic character chosen uniformly random.
- **InitCharMixed**: Each gene will be set to an alphabetic character chosen uniformly random.
- **InitPermutation**:
- **InitRange**:
- **InitPercent**:
- **InitUser**: User-defined initialization (see `Initialize` parameter).
Probability of setting 1

    OPTIONAL CONSTANT double BinInitProbBit1 = "0.5";

When InitializationType = InitBinRandom, the probability of 1 is set according to this parameter.

Low range integer for initialization

    OPTIONAL CONSTANT int IntInitRangeLow = "0";

Specifies the lower initialization range for integer strings.

High range integer for initialization

    OPTIONAL CONSTANT int IntInitRangeHigh = "ChromosomeLength";

Specifies the higher initialization range for integer strings.

Low range real for initialization

    OPTIONAL CONSTANT double RealInitRangeLow = "0.0";

Specifies the lower initialization range for real strings.

High range real for initialization

    OPTIONAL CONSTANT double RealInitRangeHigh = "1.0";

Specifies the higher initialization range for real strings.

Initialize procedure

    OPTIONAL PROCEDURE Initialize (PGAContextPtr ctx, int p, int pop);

User defined initialization procedure, required when InitializationType = InitUser.

Selection

Selection type

    TYPE SelectionTypeEnum = enum {SelectionProportional, SelectionSUS, Selectic
    OPTIONAL CONSTANT SelectionTypeEnum SelectionType = "SelectionTournament";

This parameters selects which one of the following selection schemes is used:

- SelectionProportional: proportional
- SelectionSUS: stochastic universal
- SelectionTournament: tournament
- SelectionProbTournament: probabilistic tournament. In this case the
  ProbabilisticTournamentProb can be specified.

Probabilistic tournament probability

    OPTIONAL CONSTANT double ProbabilisticTournamentProb = "0.6";

When probabilistic tournament selection is choosen, this parameter can be set in order to
specify the probability that the string that wins the tournament is selected.

Crossover

The crossover operators takes alleles from two parent strings and combines them to create two child strings.

Crossover type

```plaintext
TYPE CrossoverTypeEnum = enum {CrossoverOnePt, CrossoverTwoPt, CrossoverUniform, CrossoverUser};
OPTIONAL CONSTANT CrossoverTypeEnum CrossoverType = "CrossoverTwoPt";
```

This parameter specifies which kind of crossover will be applied.

- **CrossoverOnePt**: One point crossover
- **CrossoverTwoPt**: Two point crossover
- **CrossoverUniform**: Uniform crossover
- **CrossoverUser**: User-defined crossover

In case of selecting CrossoverUser, the crossover procedure should be also given. In case of selecting CrossoverUniform, the associate probability may also be given.

Crossover probability

```plaintext
OPTIONAL CONSTANT double CrossoverProb = "0.85";
```

This parameter corresponds to the crossover rate.

Uniform crossover probability

```plaintext
OPTIONAL CONSTANT double UniformCrossoverProb = "0.6";
```

In case of selecting CrossoverUniform crossover type, the associate probability is given by this parameter.

Crossover procedure

```plaintext
OPTIONAL PROCEDURE Crossover (PGACtxPtr ctx, int c1, int c2, int c_pop, 
Enables the user to specify his own crossover procedure.
```

Mutation

Mutation probability

```plaintext
OPTIONAL CONSTANT double MutationProb;
```

The mutation rate is the probability that a gene will undergo mutation. The default mutation probability is the reciprocal of the string length.

Mutation type

```plaintext
TYPE MutationTypeEnum = enum {MutationConstant, MutationRange, MutationUniform};
OPTIONAL CONSTANT MutationTypeEnum MutationType;
```
The type of the mutation depends on the data type. For binary-valued strings, mutation is a bit complement operation. For character-valued strings, mutation replaces one alphabetic character with another chosen uniformly. The alphabetic characters will be lower, upper, or mixed case, depending on how string were initialized. For integer-valued strings, if the strings were initialized to a permutation, the default mutation operator swaps a gene with another randomly selected gene. If the strings were initialized to a random value specified from a range, by default a gene will be replaced by a value selected uniformly random from the initialization range.

These defaults can be changed, or user-defined procedure can be written.

**Bound mutation**

```plaintext
OPTIONAL CONSTANT Boolean BoundMutation = "True";
```

Some of the integer- and real-valued mutation operators may generate allele values outside the initialization range of that gene. This parameter controls that if this happens, the allele will be reset to the lower (or upper) value of the initialization range of that gene. Setting it to False, the allele values will not be reset if they fall outside of the initialization range.

```plaintext
OPTIONAL CONSTANT int IntMutationValue = "1";
```

```plaintext
OPTIONAL CONSTANT double RealMutationValue = "0.1";
```

**Mutation function**

```plaintext
OPTIONAL FUNCTION int Mutation (PGAContextPtr ctx, int p, int pop, double mr)
```

Enables the user to specify his own mutation function. The value returned must be the number of individual mutations applied.

**Fitness**

**Fitness mapping type**

```plaintext
TYPE FitnessMappingTypeEnum = enum (FitnessRaw, FitnessRanking, FitnessNormal);
OPTIONAL CONSTANT FitnessMappingTypeEnum FitnessMappingType = "FitnessRaw";
```

This parameter specifies how the evaluation cost of an string is mapped to its fitness when maximizing. The possible choices are:

- **FitnessRaw**: identity function
- **FitnessRanking**: linear ranking
- **FitnessNormal**: linear normalization

**Ranking with MaxFitness**

```plaintext
OPTIONAL CONSTANT double MaxFitnessRank = "1.2";
```

Specifies the maximum value multiplication factor for ranking when FitnessType=FitnessNormal.
Minimization fitness mapping type

```plaintext
TYPE MinimizationFitnessTypeEnum = enum (FitnessMinCMax, FitnessMinReciprocal
OPTIONAL CONSTANT MinimizationFitnessTypeEnum MinimizationFitnessType = "FitnessMinCMax"
```

This parameter specifies how the evaluation cost of an string is mapped to its fitness when minimizing. The possible choices are:

- **FitnessMinCMax**: remaps by subtracting the worst evaluation (multiplied by MinimizationMultiplier) from each evaluation.
- **FitnessMinReciprocal**: uses the reciprocal of the evaluation function

MinimizationMultiplier

```plaintext
OPTIONAL CONSTANT double MinimizationMultiplier = "1.01";
```

Specifies the value of the multiplication factor for ranking when MinimizationFitnessType=FitnessMinCMax.

**Restart**

*Apply restart operator*

```plaintext
OPTIONAL CONSTANT Boolean ApplyRestartOperator = "false"
```

The restart operator reseeds a population from the best string. It does so by seeding the new population with the best string and generating the remaining of the population as mutated variants of the best string. This parameter is a flag to indicate if the restart operator should be used.

**Restart frequency**

```plaintext
OPTIONAL CONSTANT int RestartFrequency = "50"
```

This parameter specifies at how many iterations the restart operator will be invoked.

**RestartAlleleChangeProb**

```plaintext
OPTIONAL CONSTANT double RestartAlleleChangeProb = "0.5"
```

When creating the new strings from the best string, an individual allele will undergo mutation with the probability specified by this parameter.

**Report options**

*OnlineAnalysis*

```plaintext
OPTIONAL CONSTANT Boolean OnlineAnalysis = "false"
```

When set to `true` the library will display online analysis.

*OfflineAnalysis*
OPTIONAL CONSTANT Boolean OfflineAnalysis = "False";

When set to True the library will display offline analysis.

WorstEvaluation

OPTIONAL CONSTANT Boolean WorstEvaluation = "False";

When set to True the library will display the worst evaluation.

AverageEvaluation

OPTIONAL CONSTANT Boolean AverageEvaluation = "False";

When set to True the library will display the average evaluation.

HammingDistance

OPTIONAL CONSTANT Boolean HammingDistance = "False";

When set to True the library will display the Hamming distance.

StringItself

OPTIONAL CONSTANT Boolean StringItself = "False";

When set to True the library will display the string itself.

FrequencyValue

OPTIONAL CONSTANT int FrequencyValue = "10";

This parameter controls how often the population statistics will be printed.

EndOfGeneration

OPTIONAL PROCEDURE EndOfGeneration (PGAContextPtr ctx);

If given, this function will be called after each iteration of the GA is completed.

Miscellaneous

RandomSeed

OPTIONAL CONSTANT int RandomSeed;

If this parameter is not set, the library will initialize the random number generator based on a current time value. If it is set, the seed will be initialized with its value.

PrintContextVariable

OPTIONAL CONSTANT Boolean PrintContextVariable = "False";
When set to True the library will print a summary of the contents of the context variable before beginning the genetic algorithm.

**PrintPGAVersionNumber**

```
OPTIONAL CONSTANT Boolean PrintPGAVersionNumber = "False";
```

When set to True the library will print the version of the PgaPack library before beginning the genetic algorithm.

**PrintPGAFramVersionNumber**

```
OPTIONAL CONSTANT Boolean PrintPGAFramVersionNumber = "False";
```

When set to True the library will print the version of the PGA frame before beginning the genetic algorithm.

**PrintString**

```
OPTIONAL PROCEDURE PrintString (PGAContextPtr ctx, FILEPtr f, int p, int p0)
```

If this function is given, it will be used when printing a string is necessary. Thus, it can be used to override the default format of strings.

**Functions**

Sorry, this part is not yet documented. But meanwhile, you can take a look at the manual pages of the PgaPack library.

**More information**

Information about the PgaFrame will be keep on-line at [http://wwwlsi.upc.es/~ipetit/PgaFrame](http://wwwlsi.upc.es/~ipetit/PgaFrame) and at [http://wwwlsi.upc.es/~nlcom-it/frames/FramesRepository/PGA](http://wwwlsi.upc.es/~nlcom-it/frames/FramesRepository/PGA).

Jordi Petit i Silvestre is reachable by electronic mail as ipetit@lsi.upc.es.

**References**

   - Thomas Römke / Jordi Petit i Silvestre
   - TR-001-97 (PCPaderbora) / LSI-97-9-R (LSI Barcelona)
   - frames.ps.gz (PostScript + gzip)
2. The Frames Poster (1997)
   - Jordi Petit i Silvestre / Thomas Römke
   - LSI-97-1-T
   - poster1.ps.gz (PostScript + gzip)
   - poster1.eps.gz (EncapsulatedPostScript + gzip)
   - poster1.a1.eps.gz (DIN A1 - EncapsulatedPostScript + gzip)

   - Jordi Petit i Silvestre / Jordi Giribet Perich / Thomas Römke / Uwe Dralle
   - LSI-96-14-T
   - fit1.ps.gz (PostScript + gzip)

   - Thomas Römke / Jordi Petit i Silvestre
   - http://www.uni-paderborn.de/~alcom-it/PRreport (HTML)

   - David Levine
   - user_guide.ps (Postscript)

   - David Beasley / David R. Bull / Ralph R. Martin
   - ga_overview1.ps (Postscript)

   - David Beasley / David R. Bull / Ralph R. Martin
   - ga_overview2.ps (Postscript)

8. The Traveling Salesman Problem, A Study in Local Optimization (1993)
   - David S. Johnson / Lyle A. McGeoch
   - (Postscript)

   - Jordi Petit i Silvestre
   - http://wwwlsi.upc.es/~alcom-it/frames/FrameRepository/PGA/Html/PgaFrame.html (HTML)

This work has been partially supported by ESPRIT LTR Project no. 20244 - Alcom-IT.
ABSTRACT FRAME PGA;

DECLARATION

    /* Enumeration types */

    TYPE Boolean = enum {False, True};
    TYPE DataTypeEnum = enum {Binary, Integer, Real, Character};
    TYPE OptDirectionEnum = enum {Maximize, Minimize};
    TYPE ReplacementTypeEnum = enum {ReplaceBest, ReplaceRandom, ReplaceRandomNoRep};
    TYPE MutationCrossoverEnum = enum {MutationCrossoverOR, MutationCrossoverAND};
    TYPE CrossoverTypeEnum = enum {CrossoverOnePt, CrossoverTwoPt, CrossoverUniform, CrossoverUser};
    TYPE FitnessMappingTypeEnum = enum {FitnessRaw, FitnessRanking, FitnessNormal};
    TYPE MinimizationFitnessTypeEnum = enum {FitnessMinCMax, FitnessMinReciprocal};
    TYPE SelectionTypeEnum = enum {SelectionProportional, SelectionSUS,
                                  SelectionTournament, SelectionProbTournament};
    TYPE InitializationTypeEnum = enum {InitZero, InitBinRandom, InitCharLower, InitCharUpper, InitCharMixed,
                                          InitPermutation, InitRange, InitPercent, InitUser};
    TYPE MutationTypeEnum = enum {MutationConstant, MutationRange, MutationUniform,
                                   MutationGaussian, MutationPermutation, MutationUser};

    /* Out types (privates) */

    OUT TYPE PGAContext;
    OUT TYPE FILE;

    /* Pointer equivalent types */

    TYPE PGAContextPtr = PGAContext*;
    TYPE FILEPtr = FILE*;
TYPE charPtrPtr = char**;

/* Fundamental parameters */

CONSTANT DataTypeEnum DataType;
CONSTANT OptDirectionEnum Direction;
FUNCTION double Evaluate (PGAContextPtr ctx, int p, int pop);
FUNCTION int Preprocess (int argc, charPtrPtr argv);
OPTIONAL PROCEDURE Postprocess (int argc, charPtrPtr argv, PGAContextPtr ctx);
OPTIONAL STATEMENTUserCode();

/* Population replacement */

OPTIONAL CONSTANT int PopulationSize = "100";
OPTIONAL CONSTANT int ReplacementValue = "10";
OPTIONAL CONSTANT ReplacementTypeEnum ReplacementType = "ReplaceBest";
OPTIONAL CONSTANT MutationCrossoverEnum MutationCrossover = "MutationCrossoverOR";
OPTIONAL CONSTANT Boolean AllowDuplicates = "False";
OPTIONAL FUNCTION int DuplicateChecking (PGAContextPtr ctx, int p1, int pop1, int p2, int pop2);

/* Stopping criteria */

OPTIONAL CONSTANT Boolean UserDefinedStopRule = "False";
OPTIONAL CONSTANT Boolean StopMaxIter = "True";
OPTIONAL CONSTANT Boolean StopNoChange = "False";
OPTIONAL CONSTANT Boolean StopTooSimilar = "False";
OPTIONAL CONSTANT int MaxIterValue = "1000";
OPTIONAL CONSTANT int MaxNoChangeValue = "100";
OPTIONAL CONSTANT int MaxSimilarValue = "95";
OPTIONAL FUNCTION int StopCondition (PGAContextPtr ctx);

/* Initialization */

OPTIONAL CONSTANT InitializationTypeEnum InitializationType;
OPTIONAL CONSTANT double BinInitProbBit1 = "0.5";
OPTIONAL CONSTANT int IntInitRangeLow = "0";
OPTIONAL CONSTANT int IntInitRangeHigh = "ChromosomeLength";
OPTIONAL CONSTANT double RealInitRangeLow = "0.0";
OPTIONAL CONSTANT double RealInitRangeHigh = "1.0";
OPTIONAL PROCEDURE Initialize (PGAContextPtr ctx, int p, int pop);
/* Selection */

OPTIONAL CONSTANT SelectionTypeEnum SelectionType = "SelectionTournament";
OPTIONAL CONSTANT double ProbabilisticTournamentProb = "0.6";

/* Crossover */

OPTIONAL CONSTANT CrossoverTypeEnum CrossoverType = "CrossoverTwoPt";
OPTIONAL CONSTANT double CrossoverProb = "0.85";
OPTIONAL CONSTANT double UniformCrossoverProb = "0.5";
OPTIONAL PROCEDURE Crossover (PGAContextPtr ctx, int c1, int c2, int c_pop, int p1, int p2, int p_pop);

/* Mutation */

OPTIONAL CONSTANT double MutationProb = "0.001";
OPTIONAL CONSTANT MutationTypeEnum MutationType;
OPTIONAL CONSTANT Boolean BoundMutation = "True";
OPTIONAL CONSTANT int IntMutationValue = "1";
OPTIONAL CONSTANT double RealMutationValue = "0.1";
OPTIONAL FUNCTION int Mutation (PGAContextPtr ctx, int p, int pop, double mr);

/* Fitness */

OPTIONAL CONSTANT FitnessMappingTypeEnum FitnessMappingType = "FitnessRaw";
OPTIONAL CONSTANT double MaxFitnessRank = "1.2";
OPTIONAL CONSTANT MinimizationFitnessTypeEnum MinimizationFitnessType = "FitnessMinCMax";
OPTIONAL CONSTANT double MinimizationMultiplier = "1.01";

/* Restart */

OPTIONAL CONSTANT Boolean ApplyRestartOperator = "False";
OPTIONAL CONSTANT int RestartFrequency = "50";
OPTIONAL CONSTANT double RestartAlleleChangeProb = "0.5";

/* Report options */

OPTIONAL CONSTANT Boolean OnlineAnalysis = "False";
OPTIONAL CONSTANT Boolean OfflineAnalysis = "False";
OPTIONAL CONSTANT Boolean WorstEvaluation = "False";
OPTIONAL CONSTANT Boolean AverageEvaluation = "False";
OPTIONAL CONSTANT Boolean HammingDistance = "False";
OPTIONAL CONSTANT Boolean StringItself = "False";
OPTIONAL CONSTANT int FrequencyValue = "10";
OPTIONAL PROCEDURE EndOfGeneration (PGAContextPtr ctx);

/* Miscellaneous */

OPTIONAL CONSTANT int RandomSeed;
OPTIONAL CONSTANT Boolean PrintContextVariable = "False";
OPTIONAL CONSTANT Boolean PrintPGAVersionNumber = "False";
OPTIONAL CONSTANT Boolean PrintPGAFrameworkVersionNumber = "False";
OPTIONAL PROCEDURE PrintString (PGAContextPtr ctx, FILEPtr f, int p, int pop);

DOCUMENTATION

DOC(PGA) = URL "http://www-lsi.upc.es/~alcom-it/frames/FramesRepository/PGA/PGA.html";

END PGA.