

# GenAlg

P. Baillehache

February 9, 2020

## Contents

<b>1</b>	<b>Definitions</b>	<b>2</b>
1.1	Selection . . . . .	2
1.2	Reproduction . . . . .	2
1.3	Mutation . . . . .	2
<b>2</b>	<b>Interface</b>	<b>3</b>
<b>3</b>	<b>Code</b>	<b>12</b>
3.1	genalg.c . . . . .	12
3.2	genalg-inline.c . . . . .	43
<b>4</b>	<b>Makefile</b>	<b>56</b>
<b>5</b>	<b>Unit tests</b>	<b>56</b>
<b>6</b>	<b>Unit tests output</b>	<b>67</b>

## Introduction

GenAlg is a C library providing structures and functions implementing a Genetic Algorithm.

The genes are memorized as a VecFloat and/or VecShort. The user can defined a range of possible values for each gene. The user can define the size of the pool of entities and the size of the breeding pool. Selection, reproduction and mutation are designed to efficiently explore all the possible gene combination, and avoid local optimum. It is also possible to save and load

the GenAlg.

It uses the PBErr, PBMath and GSet libraries.

## 1 Definitions

A genetic algorithm has 3 steps. In a pool of entities it discards a given number of entities based on their ranking (given by a mean external to the algorithm). Then it replaces each of the discarded entity by a new one created from two selected entities from the non discarded one. The newly created entity's properties are a mix of these two selected entities, plus a certain amount of random modification. The detail of the implementation in GenAlg of these 3 steps (selection, reproduction and mutation) are given below.

### 1.1 Selection

The non discarded entities are called 'elite' in GenAlg. The size of the pool of elite is configurable by the user. The selection of two elite entities is simply a random selection in the pool of elites. Selection of the same elite twice is allowed.

### 1.2 Reproduction

The reproduction step copies the genes of the elite entity into the new entity. Each gene has a probability of 50% to be chosen in one or the other elite.

### 1.3 Mutation

The mutation occurs as follow. First we calculate the probability of mutation for every gene as follow:  $P = \frac{rank}{nbEntity} * (1 - \frac{1}{\sqrt{age+1}})$  where rank is the rank of the discarded entity in the pool of entities, and nbEntity is the number of entities in the pool, and age is the age of the oldest elite entity used during the reproduction step for the entity. A gene affected by a mutation according to this probability is modified as follow. The amplitude of the mutation is equal to  $1 - \frac{1}{\sqrt{age+1}}$  where age is the age of the oldest elite entity used during

the reproduction step for the entity. Then the new value of the gene is equals to  $gene + range * amp * (rnd + delta)$  where  $gene$  is the current value of the gene,  $range$  is equal to  $max_{gene} - min_{gene}$  (the difference of the maximum allowed value for this gene and its minimum value),  $amp$  is the amplitude calculated above,  $rnd$  is a random value between -0.5 and 0.5, and  $delta$  is the mutation that has been applied to this gene in the corresponding elite entity. Genes' value is kept in bounds by bouncing it on the bounds when necessary ( $gene = 2 * bound - gene$ )

To counteract inbreeding (the algorithm getting stuck into a local minimum), when the diversity level of the elite pool falls below a threshold, we also reset the  $adn$  of all the non entities and all the elite entities (except the best one) having at least one diversity level with another elite entity below the diversity level of the elite pool (set to 0.01 by default). The diversity level of the whole elite pool is calculated as follow  $Avg_{i,j} \frac{\|\vec{adn}(elite_i) - \vec{adn}(elite_j)\|}{\|\vec{bound}_{max} - \vec{bound}_{min}\|}$  where  $\vec{adn}(elite_i)$  is the genes vector of the  $i$ -th elite entity, and  $\vec{bound}_{max}$  and  $\vec{bound}_{min}$  are the vector of maximum and minimum values of the genes.

Some explanation:  $delta$  bias the mutation toward the direction that improved the result at previous step; in the pool of discarded entities high ranked ones tend to have few mutations and low ranked ones tend to have more mutation, this tends to cover any possibilities of evolution; entities newly entered in the elite pool tends to produce new entities near to them (in term of distance in the genes space), while older ones tend to produce more diverse new entities, thus the exploration of solution space occurs from the vicinity of newly better solutions toward larger areas; from the previous point, a good entity tends to create a lot of similar entity, which may lead to an elite pool saturated with very similar entities (inbreeding) from which the algorithm can't escape, this is prevented by the forced mutation of elites when the inbreeding level gets too high.

## 2 Interface

```
// ===== GENALG.H =====

#ifndef GENALG_H
#define GENALG_H

// ===== Include =====

#include <stdlib.h>
```

```

#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdbool.h>
#include "pberr.h"
#include "pbmath.h"
#include "gset.h"
#include "republish.h"

// ===== Define =====

#define GABestAdnF(that) GAAdnAdnF(GABestAdn(that))
#define GABestAdnI(that) GAAdnAdnI(GABestAdn(that))

#define GENALG_NBENTITIES 100
#define GENALG_NBELITES 20

#define GENALG_TXTOMETER_NBADNDISPLAYED 40
#define GENALG_TXTOMETER_LINE1 "Epoch #xxxxxx KTEvent #xxxxxx \n"
#define GENALG_TXTOMETER_FORMAT1 "Epoch #06lu KTEvent #06lu\n"
#define GENALG_TXTOMETER_LINE2 "Id      Age      Val\n"
#define GENALG_TXTOMETER_LINE3 "xxxxxxx  xxxxxx  +xxxxxx.xxxxxx\n"
#define GENALG_TXTOMETER_FORMAT3 "%08lu  %06lu  %+06.6f\n"
#define GENALG_TXTOMETER_LINE4 "-----\n"
#define GENALG_TXTOMETER_LINE5 "Diversity +xxx.xxxx/+xxx.xxxx \n"
#define GENALG_TXTOMETER_FORMAT5 "Diversity %+03.5f/%+03.5f \n"
#define GENALG_TXTOMETER_LINE6 "Size pool xxxxxx \n"
#define GENALG_TXTOMETER_FORMAT6 "Size pool %06d \n"

// ----- GenAlgAdn

// ===== Data structure =====

typedef struct GenAlg GenAlg;

typedef struct GenAlgAdn {
    // ID
    unsigned long _id;
    // Age
    unsigned long _age;
    // Adn for floating point value
    VecFloat* _adnF;
    // Delta Adn during mutation for floating point value
    VecFloat* _deltaAdnF;
    // Adn for integer value
    VecLong* _adnI;
    // Value
    float _val;
    // Mutability of adn for floating point value
    VecFloat* _mutabilityF;
    // Mutability of adn for integer value
    VecFloat* _mutabilityI;
} GenAlgAdn;

// ===== Functions declaration =====

// Create a new GenAlgAdn with ID 'id', 'lengthAdnF' and 'lengthAdnI'
// 'lengthAdnF' and 'lengthAdnI' must be greater than or equal to 0
GenAlgAdn* GenAlgAdnCreate(const unsigned long id, const long lengthAdnF,
    const long lengthAdnI);

// Free memory used by the GenAlgAdn 'that'

```

```

void GenAlgAdnFree(GenAlgAdn* that);

// Return the adn for floating point values of the GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
const VecFloat* GAAdnAdnF(const GenAlgAdn* const that);

// Return the delta of adn for floating point values of the
// GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
const VecFloat* GAAdnDeltaAdnF(const GenAlgAdn* const that);

// Return the adn for integer values of the GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
VecLong* GAAdnAdnI(const GenAlgAdn* const that);

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga' according to the type of the GenAlg
void GAAdnInit(const GenAlgAdn* const that, const GenAlg* ga);

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga', version used to calculate the parameters of a NeuraNet
void GAAdnInitNeuraNet(const GenAlgAdn* const that, const GenAlg* ga);

// Get the 'iGene'-th gene of the adn for floating point values of the
// GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
float GAAdnGetGeneF(const GenAlgAdn* const that, const long iGene);

// Get the delta of the 'iGene'-th gene of the adn for floating point
// values of the GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
float GAAdnGetDeltaGeneF(const GenAlgAdn* const that, const long iGene);

// Get the 'iGene'-th gene of the adn for int values of the
// GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
int GAAdnGetGeneI(const GenAlgAdn* const that, const long iGene);

// Set the 'iGene'-th gene of the adn for floating point values of the
// GenAlgAdn 'that' to 'gene'
#if BUILDMODE != 0
static inline
#endif
void GAAdnSetGeneF(GenAlgAdn* const that, const long iGene,
    const float gene);

// Set the delta of the 'iGene'-th gene of the adn for floating point
// values of the GenAlgAdn 'that' to 'delta'
#if BUILDMODE != 0
static inline

```

```

#endif
void GAAdnSetDeltaGeneF(GenAlgAdn* const that, const long iGene,
    const float delta);

// Set the 'iGene'-th gene of the adn for int values of the
// GenAlgAdn 'that' to 'gene'
#if BUILDMODE != 0
static inline
#endif
void GAAdnSetGeneI(GenAlgAdn* const that, const long iGene,
    const long gene);

// Get the id of the GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned long GAAdnGetId(const GenAlgAdn* const that);

// Get the age of the GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned long GAAdnGetAge(const GenAlgAdn* const that);

// Get the value of the GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
float GAAdnGetVal(const GenAlgAdn* const that);

// Print the information about the GenAlgAdn 'that' on the
// stream 'stream'
void GAAdnPrintln(const GenAlgAdn* const that, FILE* const stream);

// Return true if the GenAlgAdn 'that' is new, i.e. is age equals 1
// Return false
#if BUILDMODE != 0
static inline
#endif
bool GAAdnIsNew(const GenAlgAdn* const that);

// Copy the GenAlgAdn 'tho' into the GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
void GAAdnCopy(GenAlgAdn* const that, const GenAlgAdn* const tho);

// Set the mutability vectors for the GenAlgAdn 'that' to 'mutability'
#if BUILDMODE != 0
static inline
#endif
void GAAdnSetMutabilityInt(GenAlgAdn* const that,
    const VecFloat* const mutability);
#if BUILDMODE != 0
static inline
#endif
void GAAdnSetMutabilityFloat(GenAlgAdn* const that,
    const VecFloat* const mutability);

// ----- GenAlg
// ===== Data structure =====

```

```

typedef enum GenAlgType {
    genAlgTypeDefault,
    genAlgTypeNeuraNet,
    genAlgTypeNeuraNetConv
} GenAlgType;

// Data used when GenAlg is applied to a NeuraNet
typedef struct GANeuraNet {
    // Nb of input, hidden and output of the NeuraNet
    int _nbIn;
    int _nbHid;
    int _nbOut;
    long _nbBaseConv;
    long _nbBaseCellConv;
    long _nbLink;
} GANeuraNet;

typedef struct GenAlg {
    // GSet of GenAlgAdn, sortval == score so the head of the set is the
    // worst adn and the tail of the set is the best
    GSet* _adns;
    // Copy of the best adn
    GenAlgAdn* _bestAdn;
    // Type of the GenAlg
    GenAlgType _type;
    // Current epoch
    unsigned long _curEpoch;
    // Nb elite entities in population
    int _nbElites;
    // Id of the next new GenAlgAdn
    unsigned long _nextId;
    // Length of adn for floating point value
    const long _lengthAdnF;
    // Length of adn for integer value
    const long _lengthAdnI;
    // Bounds (min, max) for floating point values adn
    VecFloat2D* _boundsF;
    // Bounds (min, max) for integer values adn
    VecLong2D* _boundsI;
    // Norm of the range value for adns (optimization for diversity
    // calculation)
    float _normRangeFloat;
    float _normRangeInt;
    // Data used if the GenAlg is applied to a NeuraNet
    GANeuraNet _NNdata;
    // Number of ktevent
    unsigned long _nbKTEvent;
    // Flag to remember if we display info via a TextOMeter
    // about the population
    bool _flagTextOMeter;
    // DiversityThreshold
    float _diversityThreshold;
    // TextOMeter to display information about the population
    // If the TextOMeter is used, its content is refreshed at each call
    // of the function GAStep();
    TextOMeter* _textOMeter;
    // Nb min of adns
    int _nbMinAdn;
    // Nb max of adns
    int _nbMaxAdn;
} GenAlg;

```

```

// ===== Functions declaration =====

// Create a new GenAlg with 'nbEntities', 'nbElites', 'lengthAdnF'
// and 'lengthAdnI'
// 'nbEntities' must greater than 2
// 'nbElites' must greater than 1
// 'lengthAdnF' and 'lengthAdnI' must be greater than or equal to 0
GenAlg* GenAlgCreate(const int nbEntities, const int nbElites,
    const long lengthAdnF, const long lengthAdnI);

// Free memory used by the GenAlg 'that'
void GenAlgFree(GenAlg** that);

// Get the type of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
GenAlgType GAGetType(const GenAlg* const that);

// Set the type of the GenAlg 'that' to genAlgTypeNeuraNet, the GenAlg
// will be used with a NeuraNet having 'nbIn' inputs, 'nbHid' hidden
// values and 'nbOut' outputs
#if BUILDMODE != 0
static inline
#endif
void GASetTypeNeuraNet(GenAlg* const that, const int nbIn,
    const int nbHid, const int nbOut);

// Set the type of the GenAlg 'that' to genAlgTypeNeuraNetConv,
// the GenAlg will be used with a NeuraNet having 'nbIn' inputs,
// 'nbHid' hidden values, 'nbOut' outputs, 'nbBaseConv' bases function,
// 'nbLink' links dedicated to the convolution and 'nbBaseCellConv' bases function per cell of convolution
#if BUILDMODE != 0
static inline
#endif
void GASetTypeNeuraNetConv(GenAlg* const that, const int nbIn,
    const int nbHid, const int nbOut, const long nbBaseConv,
    const long nbBaseCellConv, const long nbLink);

// Return the GSet of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
GSet* GAAdns(const GenAlg* const that);

// Return the max nb of adns of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
int GAGetNbMaxAdn(const GenAlg* const that);

// Return the min nb of adns of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
int GAGetNbMinAdn(const GenAlg* const that);

// Set the min nb of adns of the GenAlg 'that' to 'nb'
#if BUILDMODE != 0
static inline
#endif

```



```

void GASETnbMaxAdn(GenAlg* const that, const int nb);

// Set the min nb of adns of the GenAlg 'that' to 'nb'
#if BUILDMODE != 0
static inline
#endif
void GASETnbMinAdn(GenAlg* const that, const int nb);

// Return the nb of entities of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
int GAGETnbAdns(const GenAlg* const that);

// Get the diversity threshold of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
float GAGETDiversityThreshold(const GenAlg* const that);

// Set the diversity threshold of the GenAlg 'that' to 'threshold'
#if BUILDMODE != 0
static inline
#endif
void GASETdiversityThreshold(GenAlg* const that, const float threshold);

// Return the nb of elites of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
int GAGETnbElites(const GenAlg* const that);

// Return the current epoch of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned long GAGETCurEpoch(const GenAlg* const that);

// Return the number of KTEvent of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned long GAGETnbKTEvent(const GenAlg* const that);

// Set the nb of entities of the GenAlg 'that' to 'nb'
// 'nb' must be greater than 1, if 'nb' is lower than the current nb
// of elite the number of elite is set to 'nb' - 1
void GASETnbEntities(GenAlg* const that, const int nb);

// Set the nb of elites of the GenAlg 'that' to 'nb'
// 'nb' must be greater than 0, if 'nb' is greater or equal to the
// current nb of entities the number of entities is set to 'nb' + 1
void GASETnbElites(GenAlg* const that, const int nb);

// Get the length of adn for floating point value
#if BUILDMODE != 0
static inline
#endif
long GAGETLengthAdnFloat(const GenAlg* const that);

// Get the length of adn for integer value
#if BUILDMODE != 0

```

```

static inline
#endif
long GAGetLengthAdnInt(const GenAlg* const that);

// Get the bounds for the 'iGene'-th gene of adn for floating point
// values
#if BUILDMODE != 0
static inline
#endif
const VecFloat2D* GABoundsAdnFloat(const GenAlg* const that,
    const long iGene);

// Get the bounds for the 'iGene'-th gene of adn for integer values
#if BUILDMODE != 0
static inline
#endif
const VecLong2D* GABoundsAdnInt(const GenAlg* const that,
    const long iGene);

// Set the bounds for the 'iGene'-th gene of adn for floating point
// values to a copy of 'bounds'
#if BUILDMODE != 0
static inline
#endif
void GASetBoundsAdnFloat(GenAlg* const that, const long iGene,
    const VecFloat2D* const bounds);

// Set the bounds for the 'iGene'-th gene of adn for integer values
// to a copy of 'bounds'
#if BUILDMODE != 0
static inline
#endif
void GASetBoundsAdnInt(GenAlg* const that, const long iGene,
    const VecLong2D* const bounds);

// Get the GenAlgAdn of the GenAlg 'that' currently at rank 'iRank'
#if BUILDMODE != 0
static inline
#endif
GenAlgAdn* GAAdn(const GenAlg* const that, const int iRank);

// Init the GenAlg 'that'
// Must be called after the bounds have been set
// The random generator must have been initialised before calling this
// function
void GAINit(GenAlg* const that);

// Step an epoch for the GenAlg 'that' with the current ranking of
// GenAlgAdn
void GAStep(GenAlg* const that);

// Print the information about the GenAlg 'that' on the stream 'stream'
void GAPrintln(const GenAlg* const that, FILE* const stream);

// Print a summary about the elite entities of the GenAlg 'that'
// on the stream 'stream'
void GAEliteSummaryPrintln(const GenAlg* const that,
    FILE* const stream);

// Get the diversity of the GenAlg 'that'
#if BUILDMODE != 0
static inline

```

```

#endif
float GAGetDiversity(const GenAlg* const that);

// Function which return the JSON encoding of 'that'
JSONNode* GAEncodeAsJSON(const GenAlg* const that);

// Function which decode from JSON encoding 'json' to 'that'
bool GADecodeAsJSON(GenAlg** that, const JSONNode* const json);

// Load the GenAlg 'that' from the stream 'stream'
// If the GenAlg is already allocated, it is freed before loading
// Return true in case of success, else false
bool GALoad(GenAlg** that, FILE* const stream);

// Save the GenAlg 'that' to the stream 'stream'
// If 'compact' equals true it saves in compact form, else it saves in
// readable form
// Return true in case of success, else false
bool GASave(const GenAlg* const that, FILE* const stream,
    const bool compact);

// Set the value of the GenAlgAdn 'adn' of the GenAlg 'that' to 'val'
#if BUILDMODE != 0
static inline
#endif
void GASetAdnValue(GenAlg* const that, GenAlgAdn* const adn,
    const float val);

// Update the norm of the range value for adans of the GenAlg 'that'
void GAUpdateNormRange(GenAlg* const that);

// Reset the GenAlg 'that'
// Randomize all the gene except those of the first adn
void GAKTEvent(GenAlg* const that);

// Return the best adn of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
const GenAlgAdn* GABestAdn(const GenAlg* const that);

// Return the flag memorizing if the TextOMeter is displayed for
// the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
bool GAIsTextOMeterActive(const GenAlg* const that);

// Set the flag memorizing if the TextOMeter is displayed for
// the GenAlg 'that' to 'flag'
void GASetTextOMeterFlag(GenAlg* const that, bool flag);

// ===== Polymorphism =====

// ===== static inliner =====

#if BUILDMODE != 0
#include "genalg-inline.c"
#endif

#endif

```

## 3 Code

### 3.1 genalg.c

```
// ===== GENALG.C =====

// ===== Include =====

#include "genalg.h"
#ifdef BUILDMODE == 0
#include "genalg-inline.c"
#endif

// ----- GenAlgAdn

// ===== Functions declaration =====

// Get the diversity value of 'adnA' against 'adnB'
// The diversity is equal to
float GAAdnGetDiversity(const GenAlgAdn* const adnA,
    const GenAlgAdn* const adnB, const GenAlg* const ga);

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga'
void GAAdnInitDefault(const GenAlgAdn* const that, const GenAlg* ga);

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga', version used to calculate the parameters of a NeuraNet
void GAAdnInitNeuraNet(const GenAlgAdn* const that, const GenAlg* ga);

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga', version used to calculate the parameters of a NeuraNet
// with convolution
void GAAdnInitNeuraNetConv(const GenAlgAdn* const that,
    const GenAlg* const ga);

// ===== Functions implementation =====

// Create a new GenAlgAdn with ID 'id', 'lengthAdnF' and 'lengthAdnI'
// 'lengthAdnF' and 'lengthAdnI' must be greater than or equal to 0
GenAlgAdn* GenAlgAdnCreate(const unsigned long id,
    const long lengthAdnF, const long lengthAdnI) {
#ifdef BUILDMODE == 0
    if (lengthAdnF < 0) {
        GenAlgErr->_type = PBErrTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "'lengthAdnF' is invalid (%ld>=0)",
            lengthAdnF);
        PBErrCatch(GenAlgErr);
    }
    if (lengthAdnI < 0) {
        GenAlgErr->_type = PBErrTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "'lengthAdnI' is invalid (%ld>=0)",
            lengthAdnI);
        PBErrCatch(GenAlgErr);
    }
#endif
    // Allocate memory
    GenAlgAdn* that = PBErrMalloc(GenAlgErr, sizeof(GenAlgAdn));
    // Set the properties
    that->_age = 1;
    that->_id = id;
```

```

    that->_val = 0.0;
    if (lengthAdnF > 0) {
        that->_adnF = VecFloatCreate(lengthAdnF);
        that->_deltaAdnF = VecFloatCreate(lengthAdnF);
        that->_mutabilityF = VecFloatCreate(lengthAdnF);
    } else {
        that->_adnF = NULL;
        that->_deltaAdnF = NULL;
        that->_mutabilityF = NULL;
    }
    if (lengthAdnI > 0) {
        that->_adnI = VecLongCreate(lengthAdnI);
        that->_mutabilityI = VecFloatCreate(lengthAdnI);
    } else {
        that->_adnI = NULL;
        that->_mutabilityI = NULL;
    }
    // Return the new GenAlgAdn
    return that;
}

// Free memory used by the GenAlgAdn 'that'
void GenAlgAdnFree(GenAlgAdn** that) {
    // Check the argument
    if (that == NULL || *that == NULL) return;
    // Free memory
    if ((*that)->_adnF != NULL)
        VecFree(&((*that)->_adnF));
    if ((*that)->_deltaAdnF != NULL)
        VecFree(&((*that)->_deltaAdnF));
    if ((*that)->_adnI != NULL)
        VecFree(&((*that)->_adnI));
    if ((*that)->_mutabilityF != NULL)
        VecFree(&((*that)->_mutabilityF));
    if ((*that)->_mutabilityI != NULL)
        VecFree(&((*that)->_mutabilityI));
    free(*that);
    // Set the pointer to null
    *that = NULL;
}

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga' according to the type of GenAlg
void GAAdnInit(const GenAlgAdn* const that, const GenAlg* const ga) {
    #if BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErrCatch(GenAlgErr);
        }
    #endif
    switch (GAGetType(ga)) {
        case genAlgTypeNeuraNet:
            GAAdnInitNeuraNet(that, ga);
            break;
        case genAlgTypeNeuraNetConv:
            GAAdnInitNeuraNetConv(that, ga);
            break;
        case genAlgTypeDefault:
        default:
            GAAdnInitDefault(that, ga);
    }
}

```

```

}

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga'
void GAAAdnInitDefault(const GenAlgAdn* const that,
    const GenAlg* const ga) {
    #if BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErrCatch(GenAlgErr);
        }
    #endif
    // For each floating point value gene
    for (long iGene = GAGetLengthAdnFloat(ga); iGene--;) {
        float min = VecGet(GABoundsAdnFloat(ga, iGene), 0);
        float max = VecGet(GABoundsAdnFloat(ga, iGene), 1);
        float val = min + (max - min) * rnd();
        VecSet(that->_adnF, iGene, val);
        VecSet(that->_mutabilityF, iGene, 1.0);
    }
    // For each integer value gene
    for (long iGene = GAGetLengthAdnInt(ga); iGene--;) {
        long min = VecGet(GABoundsAdnInt(ga, iGene), 0);
        long max = VecGet(GABoundsAdnInt(ga, iGene), 1);
        long val = (long)round((float)min + (float)(max - min) * rnd());
        VecSet(that->_adnI, iGene, val);
        VecSet(that->_mutabilityI, iGene, 1.0);
    }
}

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga', version used to calculate the parameters of a NeuraNet
// with convolution
void GAAAdnInitNeuraNetConv(const GenAlgAdn* const that,
    const GenAlg* const ga) {
    #if BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErrCatch(GenAlgErr);
        }
    #endif
    // For each floating point value gene
    for (long iGene = GAGetLengthAdnFloat(ga); iGene--;) {
        float min = VecGet(GABoundsAdnFloat(ga, iGene), 0);
        float max = VecGet(GABoundsAdnFloat(ga, iGene), 1);
        float val = min + (max - min) * rnd();
        VecSet(that->_adnF, iGene, val);
        VecSet(that->_mutabilityF, iGene, 1.0);
    }
}

// Initialise randomly the genes of the GenAlgAdn 'that' of the
// GenAlg 'ga', version used to calculate the parameters of a NeuraNet
void GAAAdnInitNeuraNet(const GenAlgAdn* const that, const GenAlg* ga) {
    #if BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErrCatch(GenAlgErr);
        }
    #endif
}

```

```

#endif
// Init the base functions randomly
// For each floating point value gene
for (long iGene = GAGetLengthAdnFloat(ga); iGene--;) {
    float min = VecGet(GABoundsAdnFloat(ga, iGene), 0);
    float max = VecGet(GABoundsAdnFloat(ga, iGene), 1);
    float val = min + (max - min) * rnd();
    VecSet(that->_adnF, iGene, val);
    VecSet(that->_mutabilityF, iGene, 1.0);
}
// Init the links by ensuring there is at least one link reaching
// each output and use inputs as start of the initial links
// For each integer value gene
int shiftOut = ga->_NNdata._nbIn + ga->_NNdata._nbHid;
for (long iGene = GAGetLengthAdnInt(ga); iGene--;) {
    VecSet(that->_adnI, iGene, -1);
    VecSet(that->_mutabilityI, iGene, 1.0);
}
for (int iOut = 0; iOut < ga->_NNdata._nbOut; ++iOut) {
    // The base function is randomly choosen but can't be an
    // inactive link
    long min = 0;
    long max = VecGet(GABoundsAdnInt(ga, iOut * 3), 1);
    long val = (long)round((float)min + (float)(max - min) * rnd());
    VecSet(that->_adnI, iOut * 3, val);
    // The start of the link is randomly choosen amongst inputs
    min = 0;
    max = ga->_NNdata._nbIn - 1;
    val = (long)round((float)min + (float)(max - min) * rnd());
    VecSet(that->_adnI, iOut * 3 + 1, val);
    // The end of the link is choosen sequentially amongst outputs
    VecSet(that->_adnI, iOut * 3 + 2, iOut + shiftOut);
}
}

// Print the information about the GenAlgAdn 'that' on the
// stream 'stream'
void GAAdnPrintln(const GenAlgAdn* const that, FILE* const stream) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
    if (stream == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'stream' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    fprintf(stream, "id:%lu age:%lu", GAAdnGetId(that), GAAdnGetAge(that));
    fprintf(stream, "\n");
    fprintf(stream, "  adnF:");
    if (GAAdnAdnF(that) != NULL)
        VecFloatPrint(GAAdnAdnF(that), stream, 6);
    else
        fprintf(stream, "<null>");
    fprintf(stream, "\n");
    fprintf(stream, "  deltaAdnF:");
    if (GAAdnAdnF(that) != NULL)
        VecFloatPrint(GAAdnDeltaAdnF(that), stream, 6);
    else

```

```

        fprintf(stream, "<null>");
    fprintf(stream, "\n");
    fprintf(stream, "  adnI:");
    if (GAAdnAdnI(that) != NULL)
        VecPrint(GAAdnAdnI(that), stream);
    else
        fprintf(stream, "<null>");
    fprintf(stream, "\n");
}

// ----- GenAlg

// ===== Functions declaration =====

// Select the rank of two parents for the SRM algorithm
// Return the ranks in 'parents', with parents[0] <= parents[1]
void GASelectParents(const GenAlg* const that, int* const parents);

// Set the genes of the entity at rank 'iChild' as a 50/50 mix of the
// genes of entities at ranks 'parents[0]' and 'parents[1]'
void GAReproduction(GenAlg* const that, const int* const parents,
    const int iChild);

// Set the genes of the entity at rank 'iChild' as a 50/50 mix of the
// genes of entities at ranks 'parents[0]' and 'parents[1]'
void GAReproductionDefault(GenAlg* const that,
    const int* const parents, const int iChild);

// Set the genes of the adn at rank 'iChild' as a 50/50 mix of the
// genes of adns at ranks 'parents[0]' and 'parents[1]'
// This version is optimised to calculate the parameters of a NeuraNet
// with convolution by inheriting whole bases from parents
void GAReproductionNeuraNetConv(GenAlg* const that,
    const int* const parents, const int iChild);

// Set the genes of the entity at rank 'iChild' as a 50/50 mix of the
// genes of entities at ranks 'parents[0]' and 'parents[1]'
// This version is optimised to calculate the parameters of a NeuraNet
// by inheriting whole bases and links from parents
void GAReproductionNeuraNet(GenAlg* const that,
    const int* const parents, const int iChild);

// Router toward the appropriate Mute function according to the type
// of GenAlg
void GAMute(GenAlg* const that, const int* const parents,
    const int iChild);

// Mute the genes of the entity at rank 'iChild'
void GAMuteDefault(GenAlg* const that, const int* const parents,
    const int iChild);

// Mute the genes of the entity at rank 'iChild'
// This version is optimised to calculate the parameters of a NeuraNet
// by ensuring coherence in links: outputs have at least one link
// and there is no dead link
void GAMuteNeuraNet(GenAlg* const that, const int* const parents,
    const int iChild);

// Mute the genes of the entity at rank 'iChild'
// This version is optimised to calculate the parameters of a NeuraNet
// with convolution by muting bases function per cell
void GAMuteNeuraNetConv(GenAlg* const that, const int* const parents,

```



```

    const int iChild);

// Refresh the content of the TextOMeter attached to the GenAlg 'that'
void GAUpdateTextOMeter(const GenAlg* const that);

// ===== Functions implementation =====

// Create a new GenAlg with 'nbEntities', 'nbElites', 'lengthAdnF'
// and 'lengthAdnI'
// 'nbEntities' must be greater than 2
// 'nbElites' must be greater than 1
// 'lengthAdnF' and 'lengthAdnI' must be greater than or equal to 0
GenAlg* GenAlgCreate(const int nbEntities, const int nbElites,
    const long lengthAdnF, const long lengthAdnI) {
    // Allocate memory
    GenAlg* that = PBErrMalloc(GenAlgErr, sizeof(GenAlg));
    // Set the properties
    that->_type = genAlgTypeDefault;
    that->_adns = GSetCreate();
    that->_curEpoch = 0;
    that->_nbKTEvent = 0;
    that->_flagTextOMeter = false;
    that->_diversityThreshold = 0.01;
    that->_textOMeter = NULL;
    that->_nbMinAdn = nbEntities;
    that->_nbMaxAdn = nbEntities;
    that->_bestAdn = GenAlgAdnCreate(0, lengthAdnF, lengthAdnI);
    *(long*)&(that->_lengthAdnF) = lengthAdnF;
    *(long*)&(that->_lengthAdnI) = lengthAdnI;
    if (lengthAdnF > 0) {
        that->_boundsF =
            PBErrMalloc(GenAlgErr, sizeof(VecFloat2D) * lengthAdnF);
        for (long iGene = lengthAdnF; iGene--;)
            that->_boundsF[iGene] = VecFloatCreateStatic2D();
    } else
        that->_boundsF = NULL;
    if (lengthAdnI > 0) {
        that->_boundsI =
            PBErrMalloc(GenAlgErr, sizeof(VecLong2D) * lengthAdnI);
        for (long iGene = lengthAdnI; iGene--;)
            that->_boundsI[iGene] = VecLongCreateStatic2D();
    } else
        that->_boundsI = NULL;
    that->_normRangeFloat = 1.0;
    that->_normRangeInt = 1.0;
    that->_nbElites = 0;
    that->_nextId = 0;
    GSetNbEntities(that, nbEntities);
    GSetNbElites(that, nbElites);
    // Return the new GenAlg
    return that;
}

// Free memory used by the GenAlg 'that'
void GenAlgFree(GenAlg** that) {
    // Check the argument
    if (that == NULL || *that == NULL) return;
    // Free memory
    GSetIterForward iter = GSetIterForwardCreateStatic(GAAdns(*that));
    do {
        GenAlgAdn* gaEnt = GSetIterGet(&iter);
        GenAlgAdnFree(&gaEnt);
    } while (iter);
}

```

```

    } while (GSetIterStep(&iter));
    GSetFree(&((*that)->_adns));
    if ((*that)->_boundsF != NULL)
        free((*that)->_boundsF);
    if ((*that)->_boundsI != NULL)
        free((*that)->_boundsI);
    GenAlgAdnFree(&((*that)->_bestAdn));
    if ((*that)->_textOMeter != NULL) {
        TextOMeterFree(&((*that)->_textOMeter));
    }
    free(*that);
    // Set the pointer to null
    *that = NULL;
}

// Set the nb of entities of the GenAlg 'that' to 'nb'
// 'nb' must be greater than 1, if 'nb' is lower than the current nb
// of elite the number of elite is set to 'nb' - 1
void GASetNbEntities(GenAlg* const that, const int nb) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PErrCatch(GenAlgErr);
    }
    if (nb <= 1) {
        GenAlgErr->_type = PErrTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "'nb' is invalid (%d>1)", nb);
        PErrCatch(GenAlgErr);
    }
#endif
    while (GSetNbElem(GAAdns(that)) > nb) {
        GenAlgAdn* gaEnt = GSetPop(GAAdns(that));
        GenAlgAdnFree(&gaEnt);
    }
    while (GSetNbElem(GAAdns(that)) < nb) {
        GenAlgAdn* ent = GenAlgAdnCreate(that->_nextId++,
            GAGetLengthAdnFloat(that), GAGetLengthAdnInt(that));
        GSetPush(GAAdns(that), ent);
    }
    if (GAGetNbElites(that) >= nb)
        GASetNbElites(that, nb - 1);
}

// Set the nb of elites of the GenAlg 'that' to 'nb'
// 'nb' must be greater than 0, if 'nb' is greater or equal to the
// current nb of entities the number of entities is set to 'nb' + 1
void GASetNbElites(GenAlg* const that, const int nb) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PErrCatch(GenAlgErr);
    }
    if (nb <= 1) {
        GenAlgErr->_type = PErrTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "'nb' is invalid (%d>1)", nb);
        PErrCatch(GenAlgErr);
    }
#endif
    if (GAGetNbAdns(that) <= nb)
        GASetNbEntities(that, nb + 1);
}

```

```

    that->_nbElites = nb;
}

// Init the GenAlg 'that'
// Must be called after the bounds have been set
// The random generator must have been initialised before calling this
// function
void GAlnit(GenAlg* const that) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PErrCatch(GenAlgErr);
    }
#endif
    // For each adn
    GSetIterForward iter = GSetIterForwardCreateStatic(GAAdns(that));
    do {
        // Get the adn
        GenAlgAdn* adn = GSetIterGet(&iter);
        // Initialise randomly the genes of the adn
        GAAdnInit(adn, that);
    } while (GSetIterStep(&iter));
    GAAdnCopy(that->_bestAdn, GAAdn(that, 0));
}

// Reset the GenAlg 'that'
// Randomize all the gene except those of the best adn
void GAKTEvent(GenAlg* const that) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PErrCatch(GenAlgErr);
    }
#endif
    // No KTEvent on the first epoch
    if (GAGetCurEpoch(that) == 0)
        return;
    // Get the diversity level
    //float diversity = GAGetDiversity(that);
    // Loop until the diversity of the elites is sufficient
    int nbKTEvent = 0;
    int nbMaxLoop =
        (int)round((float)GAGetNbAdns(that) / (float)GAGetNbElites(that));
    do {
        nbKTEvent = 0;
        // For each pair of adn
        for (int iAdn = GAGetNbElites(that) - 1; iAdn >= 1; --iAdn) {
            for (int jAdn = iAdn - 1; jAdn >= 0; --jAdn) {
                // Get the diversity of this pair
                float div = fabs(GAAdnGetVal(GAAdn(that, iAdn)) -
                    GAAdnGetVal(GAAdn(that, jAdn)));
                // If it's below the diversity threshold
                if (div <= GAGetDiversityThreshold(that)) {
                    GenAlgAdn* adn = GAAdn(that, iAdn);
                    GAAdnInit(adn, that);
                    adn->_age = 1;
                    adn->_id = (that->_nextId)++;
                    GAsSetAdnValue(that, adn,
                        GAAdnGetVal(GAAdn(that, GAGetNbAdns(that) - 1)));
                    jAdn = 0;
                }
            }
        }
    } while (nbKTEvent < nbMaxLoop);
}

```

```

        ++nbKTEvent;
    }
}
}
// If their has been KTEvent
if (nbKTEvent > 0) {
    // We need to sort the adns
    GSetSort(GAAdns(that));
    // Memorize the total number of KTEvent
    that->_nbKTEvent += nbKTEvent;
}
--nbMaxLoop;
} while (nbKTEvent > 0 && nbMaxLoop > 0);
// Calculate a threshold for the age of the best elite
unsigned int th = (unsigned int)(GAGetNbMaxAdn(that) - GAGetNbAdns(that) + GAGetNbMinAdn(that));
// Get the diversity relatively to the best of all
float div = fabs(GAAdnGetVal(GAAdn(that, 0)) -
    GAAdnGetVal(GABestAdn(that)));
// If it's below the diversity threshold or the it's older than
// the threshold
if (div <= GAGetDiversityThreshold(that) ||
    GAAdnGetAge(GAAdn(that, 0)) >= th) {
    GenAlgAdn* adn = GAAdn(that, 0);
    GAAdnInit(adn, that);
    adn->_age = 1;
    adn->_id = (that->_nextId)++;
    GASetAdnValue(that, adn,
        GAAdnGetVal(GAAdn(that, GAGetNbAdns(that) - 1)));
    // We need to sort the adns
    GSetSort(GAAdns(that));
    // Memorize the total number of KTEvent
    that->_nbKTEvent += 1;
}
}

// Step an epoch for the GenAlg 'that' with the current ranking of
// GenAlgAdn
void GASep(GenAlg* const that) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    // Selection, Reproduction, Mutation
    // Ensure the set of adns is sorted
    GSetSort(GAAdns(that));
    // Variable to memorize if there has been improvement
    bool flagImprov = false;
    // Update the best adn if necessary
    if (that->_curEpoch == 1) {
        GAAdnCopy(that->_bestAdn, GAAdn(that, 0));
        that->_bestAdn->_age = that->_curEpoch + 1;
    } else {
        if (GAAdnGetVal(GAAdn(that, 0)) > GAAdnGetVal(GABestAdn(that))) {
            GAAdnCopy(that->_bestAdn, GAAdn(that, 0));
            that->_bestAdn->_age = that->_curEpoch + 1;
            flagImprov = true;
        }
    }
}
// Ensure diversity level

```

```

GAKTEvent(that);
// Refresh the TextOMeter if necessary
if (that->_flagTextOMeter) {
    GAUpdateTextOMeter(that);
}
// Resize the population according to the improvement
if (that->_curEpoch > 1) {
    if (flagImprov) {
        GASetNbEntities(that,
            MAX(GAGetNbMinAdn(that), GAGetNbAdns(that) / 2));
    } else {
        GASetNbEntities(that,
            MIN(GAGetNbMaxAdn(that), 2 * GAGetNbAdns(that)));
    }
}

/**/ Get the diversity level
float diversity = GAGetDiversity(that);
// Correct the diversity level with the age of the best adn
//diversity *=
//1.0 - fsquare((float)(GAAdnGetAge(GAAdn(that, 0))) / 1000.0);
// If the diversity level is too low
if (that->_curEpoch > 1 &&
    (diversity < GAGetDiversityThreshold(that) ||
    GAAdnGetAge(GAAdn(that, 0)) > 200)) {
    // Renew diversity by applying a KT event (in memory of
    // chickens' grand pa and grand ma)
    GAKTEvent(that);
// Else, the diversity level is ok
} else { */
    // For each adn which is an elite
    for (int iAdn = 0; iAdn < GAGetNbElites(that); ++iAdn) {
        // Increment age
        ++(GAAdn(that, iAdn)->_age);
    }
    // For each adn which is not an elite
    for (int iAdn = GAGetNbElites(that); iAdn < GAGetNbAdns(that);
        ++iAdn) {
        // Declare a variable to memorize the parents
        int parents[2];
        // Select two parents for this adn
        GASelectParents(that, parents);
        // Set the genes of the adn as a 50/50 mix of parents' genes
        GAReproduction(that, parents, iAdn);
        // Mute the genes of the adn
        GAMute(that, parents, iAdn);
    }
}
// Increment the number of epochs
++(that->_curEpoch);
}

// Select the rank of two parents for the SRM algorithm
// Return the ranks in 'parents', with parents[0] <= parents[1]
void GASelectParents(const GenAlg* const that, int* const parents) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
    if (parents == NULL) {

```

```

        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'parents' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    // Declare a variable to memorize the parents' rank
    int p[2];
    do {
        for (int i = 2; i--;)
            // p[i] below may be equal to the rank of the highest non elite
            // adn, but it's not a problem so leave it and let's call that
            // the Hawking radiation of this function in memory of this great
            // man.
            p[i] = (int)floor(rnd() * (float)GAGetNbElites(that)) - 1;
    } while (p[0] == p[1]);
    // Memorize the sorted parents' rank
    if (p[0] < p[1]) {
        parents[0] = p[0];
        parents[1] = p[1];
    } else {
        parents[0] = p[1];
        parents[1] = p[0];
    }
}

// Set the genes of the adn at rank 'iChild' as a 50/50 mix of the
// genes of adns at ranks 'parents[0]' and 'parents[1]'
void GAReproduction(GenAlg* const that,
    const int* const parents, const int iChild) {
    #if BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErrCatch(GenAlgErr);
        }
        if (parents == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'parents' is null");
            PBErrCatch(GenAlgErr);
        }
        if (iChild < 0 || iChild >= GAGetNbAdns(that)) {
            GenAlgErr->_type = PBErrTypeInvalidArg;
            sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<=%d)",
                iChild, GAGetNbAdns(that));
            PBErrCatch(GenAlgErr);
        }
    #endif
    switch (GAGetType(that)) {
        case genAlgTypeNeuraNet:
            GAReproductionNeuraNet(that, parents, iChild);
            break;
        case genAlgTypeNeuraNetConv:
            GAReproductionNeuraNetConv(that, parents, iChild);
            break;
        case genAlgTypeDefault:
        default:
            GAReproductionDefault(that, parents, iChild);
    }
}

// Set the genes of the adn at rank 'iChild' as a 50/50 mix of the
// genes of adns at ranks 'parents[0]' and 'parents[1]'

```

```

// This version is optimised to calculate the parameters of a NeuraNet
// by inheriting whole bases and links from parents
void GAReproductionNeuraNet(GenAlg* const that,
    const int* const parents, const int iChild) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
    if (parents == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'parents' is null");
        PBErrCatch(GenAlgErr);
    }
    if (iChild < 0 || iChild >= GAGetNbAdns(that)) {
        GenAlgErr->_type = PBErrTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<=%d)",
            iChild, GAGetNbAdns(that));
        PBErrCatch(GenAlgErr);
    }
#endif
    // Get the parents and child
    GenAlgAdn* parentA = GAAdn(that, parents[0]);
    GenAlgAdn* parentB = GAAdn(that, parents[1]);
    GenAlgAdn* child = GAAdn(that, iChild);
    // For each gene of the adn for floating point value
    for (long iGene = 0; iGene < GAGetLengthAdnFloat(that); iGene += 3) {
        // Get the gene from one parent or the other with equal
        // probabability
        if (rnd() < 0.5) {
            for (long jGene = 3; jGene--;) {
                VecSet(child->_adnF, iGene + jGene,
                    VecGet(parentA->_adnF, iGene + jGene));
                VecSet(child->_deltaAdnF, iGene + jGene,
                    VecGet(parentA->_deltaAdnF, iGene + jGene));
            }
        } else {
            for (long jGene = 3; jGene--;) {
                VecSet(child->_adnF, iGene + jGene,
                    VecGet(parentB->_adnF, iGene + jGene));
                VecSet(child->_deltaAdnF, iGene + jGene,
                    VecGet(parentB->_deltaAdnF, iGene + jGene));
            }
        }
    }
    // For each gene of the adn for int value
    for (long iGene = 0; iGene < GAGetLengthAdnInt(that); iGene += 3) {
        // Get the gene from one parent or the other with equal probabability
        if (rnd() < 0.5) {
            for (long jGene = 3; jGene--;) {
                VecSet(child->_adnI, iGene + jGene,
                    VecGet(parentA->_adnI, iGene + jGene));
            }
        } else {
            for (long jGene = 3; jGene--;) {
                VecSet(child->_adnI, iGene + jGene,
                    VecGet(parentB->_adnI, iGene + jGene));
            }
        }
    }
    // Reset the age of the child
    child->_age = 1;
    // Set the id of the child

```

```

    child->_id = (that->_nextId)++;
}

// Set the genes of the adn at rank 'iChild' as a 50/50 mix of the
// genes of adns at ranks 'parents[0]' and 'parents[1]'
// This version is optimised to calculate the parameters of a NeuraNet
// with convolution by inheriting whole bases from parents
void GAReproductionNeuraNetConv(GenAlg* const that,
    const int* const parents, const int iChild) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
    if (parents == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'parents' is null");
        PBErrCatch(GenAlgErr);
    }
    if (iChild < 0 || iChild >= GAGetNbAdns(that)) {
        GenAlgErr->_type = PBErrTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<=%d)",
            iChild, GAGetNbAdns(that));
        PBErrCatch(GenAlgErr);
    }
#endif
    // Get the parents and child
    GenAlgAdn* parentA = GAAdn(that, parents[0]);
    GenAlgAdn* parentB = GAAdn(that, parents[1]);
    GenAlgAdn* child = GAAdn(that, iChild);
    // For each gene of the adn for floating point value of convolution
    // base functions
    for (long iGene = 0;
        iGene < that->_NNdata._nbBaseConv * 3;
        iGene += that->_NNdata._nbBaseCellConv * 3) {
        // Get the gene from one parent or the other with equal probability
        if (rnd() < 0.5) {
            for (long jGene = that->_NNdata._nbBaseCellConv * 3;
                jGene--;) {
                VecSet(child->_adnF, iGene + jGene,
                    VecGet(parentA->_adnF, iGene + jGene));
                VecSet(child->_deltaAdnF, iGene + jGene,
                    VecGet(parentA->_deltaAdnF, iGene + jGene));
            }
        } else {
            for (long jGene = that->_NNdata._nbBaseCellConv * 3;
                jGene--;) {
                VecSet(child->_adnF, iGene + jGene,
                    VecGet(parentB->_adnF, iGene + jGene));
                VecSet(child->_deltaAdnF, iGene + jGene,
                    VecGet(parentB->_deltaAdnF, iGene + jGene));
            }
        }
    }
    // For each gene of the adn for floating point value of convolution
    // base functions
    for (long iGene = that->_NNdata._nbBaseConv * 3;
        iGene < GAGetLengthAdnFloat(that); iGene += 3) {
        // Get the gene from one parent or the other with equal probability
        if (rnd() < 0.5) {
            for (long jGene = 3; --jGene;) {

```



```

        VecSet(child->_adnF, iGene + jGene,
            VecGet(parentA->_adnF, iGene + jGene));
        VecSet(child->_deltaAdnF, iGene + jGene,
            VecGet(parentA->_deltaAdnF, iGene + jGene));
    }
} else {
    for (long jGene = 3; --jGene;) {
        VecSet(child->_adnF, iGene + jGene,
            VecGet(parentB->_adnF, iGene + jGene));
        VecSet(child->_deltaAdnF, iGene + jGene,
            VecGet(parentB->_deltaAdnF, iGene + jGene));
    }
}
}
// Reset the age of the child
child->_age = 1;
// Set the id of the child
child->_id = (that->_nextId)++;
}

// Set the genes of the adn at rank 'iChild' as a 50/50 mix of the
// genes of adns at ranks 'parents[0]' and 'parents[1]'
void GAReproductionDefault(GenAlg* const that,
    const int* const parents, const int iChild) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
    if (parents == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'parents' is null");
        PBErrCatch(GenAlgErr);
    }
    if (iChild < 0 || iChild >= GAGetNbAdns(that)) {
        GenAlgErr->_type = PBErrTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<=%d)",
            iChild, GAGetNbAdns(that));
        PBErrCatch(GenAlgErr);
    }
#endif
// Get the parents and child
GenAlgAdn* parentA = GAAAdn(that, parents[0]);
GenAlgAdn* parentB = GAAAdn(that, parents[1]);
GenAlgAdn* child = GAAAdn(that, iChild);
// For each gene of the adn for floating point value
for (long iGene = GAGetLengthAdnFloat(that); iGene--;) {
    // Get the gene from one parent or the other with equal probability
    if (rnd() < 0.5) {
        VecSet(child->_adnF, iGene, VecGet(parentA->_adnF, iGene));
        VecSet(child->_deltaAdnF, iGene,
            VecGet(parentA->_deltaAdnF, iGene));
    } else {
        VecSet(child->_adnF, iGene, VecGet(parentB->_adnF, iGene));
        VecSet(child->_deltaAdnF, iGene,
            VecGet(parentB->_deltaAdnF, iGene));
    }
}
}
// For each gene of the adn for int value
for (long iGene = GAGetLengthAdnInt(that); iGene--;) {
    // Get the gene from one parent or the other with equal probability

```

```

        if (rnd() < 0.5)
            VecSet(child->_adnI, iGene, VecGet(parentA->_adnI, iGene));
        else
            VecSet(child->_adnI, iGene, VecGet(parentB->_adnI, iGene));
    }
    // Reset the age of the child
    child->_age = 1;
    // Set the id of the child
    child->_id = (that->_nextId)++;
}

// Router toward the appropriate Mute function according to the type
// of GenAlg
void GAMute(GenAlg* const that, const int* const parents,
            const int iChild) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
    if (parents == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'parents' is null");
        PBErrCatch(GenAlgErr);
    }
    if (iChild < 0 || iChild >= GAGetNbAdns(that)) {
        GenAlgErr->_type = PBErrTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<=%d)",
                iChild, GAGetNbAdns(that));
        PBErrCatch(GenAlgErr);
    }
#endif
    switch (GAGetType(that)) {
        case genAlgTypeNeuraNet:
            GAMuteNeuraNet(that, parents, iChild);
            break;
        case genAlgTypeNeuraNetConv:
            GAMuteNeuraNetConv(that, parents, iChild);
            break;
        case genAlgTypeDefault:
        default:
            GAMuteDefault(that, parents, iChild);
    }
}

// Mute the genes of the entity at rank 'iChild'
// This version is optimised to calculate the parameters of a NeuraNet
// by ensuring coherence in links: outputs have at least one link
// and there is no dead link
void GAMuteNeuraNet(GenAlg* const that, const int* const parents,
                    const int iChild) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
    if (parents == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'parents' is null");
        PBErrCatch(GenAlgErr);
    }

```

```

}
if (iChild < 0 || iChild >= GAGetNbAdns(that)) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<%d)",
        iChild, GAGetNbAdns(that));
    PBErrCatch(GenAlgErr);
}
#endif
// Get the first parent and child
GenAlgAdn* parentA = GAAdn(that, parents[0]);
GenAlgAdn* child = GAAdn(that, iChild);
// Get the proba and amplitude of mutation
float probMute = sqrt(((float)iChild) / ((float)GAGetNbAdns(that)));
float amp = 1.0 - sqrt(1.0 / (float)(parentA->_age + 1));
probMute /= (float)(GAGetLengthAdnInt(that));
probMute += (float)(parentA->_age) / 10000.0;
// Ensure the proba is not null
if (probMute < PBMATH_EPSILON)
    probMute = PBMATH_EPSILON;
// Declare a variable to memorize if there has been mutation
bool hasMuted = false;
// Declare a variable to memorize the used values amongst input and
// hidden
long nbMaxUsedVal = that->_NNdata._nbIn + that->_NNdata._nbHid;
char* isUsed = PBErrMalloc(GenAlgErr, sizeof(char) * nbMaxUsedVal);
// Loop until there has been at least one mutation
do {
    // Reset the used values
    memset(isUsed, 0, sizeof(char) * nbMaxUsedVal);
    memset(isUsed, 1, sizeof(char) * that->_NNdata._nbIn);
    // For each gene of the adn for int value (links definitions)
    for (long iGene = 0; iGene < GAGetLengthAdnInt(that); iGene += 3) {
        // If the link mutes
        if (rnd() < probMute) {
            hasMuted = true;
            // If this link is currently inactivated
            if (GAAdnGetGeneI(child, iGene) == -1) {
                // Base function
                long iBase = (int)round((float)iGene / 3.0);
                GAAdnSetGeneI(child, iGene, iBase);
                // Input
                long min =
                    VecGet(GABoundsAdnInt(that, iGene + 1), 0);
                long max =
                    VecGet(GABoundsAdnInt(that, iGene + 1), 1);
                long val = min;
                // Ensure the input is a used value
                do {
                    val = (long)round((float)min +
                        (float)(max - min) * rnd());
                } while (isUsed[val] == 0);
                GAAdnSetGeneI(child, iGene + 1, val);
                // Output
                min = MAX(val, VecGet(GABoundsAdnInt(that, iGene + 2), 0));
                max = VecGet(GABoundsAdnInt(that, iGene + 2), 1);
                val = (long)round((float)min + (float)(max - min) * rnd());
                GAAdnSetGeneI(child, iGene + 2, val);
                if (val < nbMaxUsedVal)
                    isUsed[val] = 1;
            }
            // Else, this link is currently activated
        } else {
            // Choose between inactivation or mutation

```

```

if (rnd() < 0.5) {
    // Inactivate the link
    GAAdnSetGeneI(child, iGene, -1);
} else {
    // Input
    long min =
        VecGet(GABoundsAdnInt(that, iGene + 1), 0);
    long max =
        VecGet(GABoundsAdnInt(that, iGene + 1), 1);
    long val = min;
    // Ensure the input is a used value
    do {
        val = (long)round((float)min +
            (float)(max - min) * rnd());
    } while (isUsed[val] == 0);
    GAAdnSetGeneI(child, iGene + 1, val);
    // Output
    min = MAX(val, VecGet(GABoundsAdnInt(that, iGene + 2), 0));
    max = VecGet(GABoundsAdnInt(that, iGene + 2), 1);
    val = (long)round((float)min + (float)(max - min) * rnd());
    GAAdnSetGeneI(child, iGene + 2, val);
    if (val < nbMaxUsedVal)
        isUsed[val] = 1;
}
}
}
// Get the index of the base function
long baseFun = GAAdnGetGeneI(child, iGene);
// If the link is active
if (baseFun != -1) {
    // If the associated base function mutes
    if (rnd() < probbMute) {
        long baseFunGene = baseFun * 3;
        for (long jGene = 3; jGene--;) {
            // Get the bounds
            const VecFloat2D* const bounds =
                GABoundsAdnFloat(that, baseFunGene + jGene);
            // Declare a variable to memorize the previous value
            // of the gene
            float prevVal = GAAdnGetGeneF(child, baseFunGene + jGene);
            // Apply the mutation
            GAAdnSetGeneF(child, baseFunGene + jGene,
                GAAdnGetGeneF(child, baseFunGene + jGene) +
                (VecGet(bounds, 1) - VecGet(bounds, 0)) * amp *
                //VecGet(parentA->_mutabilityF, baseFun * 3) *
                (rnd() - 0.5) +
                GAAdnGetDeltaGeneF(child, baseFunGene + jGene));
            // Keep the gene value in bounds
            while (GAAdnGetGeneF(child, baseFunGene + jGene) <
                VecGet(bounds, 0) ||
                GAAdnGetGeneF(child, baseFunGene + jGene) >
                VecGet(bounds, 1)) {
                if (GAAdnGetGeneF(child, baseFunGene + jGene) >
                    VecGet(bounds, 1))
                    GAAdnSetGeneF(child, baseFunGene + jGene,
                        2.0 * VecGet(bounds, 1) -
                        GAAdnGetGeneF(child, baseFunGene + jGene));
                else if (GAAdnGetGeneF(child, baseFunGene + jGene) <
                    VecGet(bounds, 0))
                    GAAdnSetGeneF(child, baseFunGene + jGene,
                        2.0 * VecGet(bounds, 0) -
                        GAAdnGetGeneF(child, baseFunGene + jGene));
            }
        }
    }
}

```

```

    }
    // Update the deltaAdn
    GAAdnSetDeltaGeneF(child, baseFunGene + jGene,
        GAAdnGetGeneF(child, baseFunGene + jGene) - prevVal);
    }
}
}
} while (hasMuted == false);
free(isUsed);
}

// Mute the genes of the entity at rank 'iChild'
void GAMuteDefault(GenAlg* const that, const int* const parents,
    const int iChild) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
    if (parents == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'parents' is null");
        PBErrCatch(GenAlgErr);
    }
    if (iChild < 0 || iChild >= GAGetNbAdns(that)) {
        GenAlgErr->_type = PBErrTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<=%d)",
            iChild, GAGetNbAdns(that));
        PBErrCatch(GenAlgErr);
    }
#endif
    // Get the first parent and child
    GenAlgAdn* parentA = GAAdn(that, parents[0]);
    GenAlgAdn* child = GAAdn(that, iChild);
    // Get the proba amplitude of mutation
    float probMute = sqrt(((float)iChild) / ((float)GAGetNbAdns(that)));
    float amp = 1.0 - sqrt(1.0 / (float)(parentA->_age));
    probMute /= (float)(MAX(GAGetLengthAdnInt(that),
        GAGetLengthAdnFloat(that)));
    probMute += (float)(parentA->_age) / 10000.0;
    if (probMute < PBMATH_EPSILON)
        probMute = PBMATH_EPSILON;
    bool hasMuted = false;
    do {
        // For each gene of the adn for floating point value
        for (long iGene = GAGetLengthAdnFloat(that); iGene--;) {
            // If this gene mutes
            if (rnd() < probMute) {
                hasMuted = true;
                // Get the bounds
                const VecFloat2D* const bounds = GABoundsAdnFloat(that, iGene);
                // Declare a variable to memorize the previous value of the gene
                float prevVal = GAAdnGetGeneF(child, iGene);
                // Apply the mutation
                GAAdnSetGeneF(child, iGene, GAAdnGetGeneF(child, iGene) +
                    (VecGet(bounds, 1) - VecGet(bounds, 0)) * amp *
                    (rnd() - 0.5) + GAAdnGetDeltaGeneF(child, iGene));
                // Keep the gene value in bounds
                while (GAAdnGetGeneF(child, iGene) < VecGet(bounds, 0) ||
                    GAAdnGetGeneF(child, iGene) > VecGet(bounds, 1)) {

```

```

        if (GAAdnGetGeneF(child, iGene) > VecGet(bounds, 1))
            GAAdnSetGeneF(child, iGene,
                2.0 * VecGet(bounds, 1) - GAAdnGetGeneF(child, iGene));
        else if (GAAdnGetGeneF(child, iGene) < VecGet(bounds, 0))
            GAAdnSetGeneF(child, iGene,
                2.0 * VecGet(bounds, 0) - GAAdnGetGeneF(child, iGene));
    }
    // Update the deltaAdn
    GAAdnSetDeltaGeneF(child, iGene,
        GAAdnGetGeneF(child, iGene) - prevVal);
}
}
// For each gene of the adn for int value
for (long iGene = GAGetLengthAdnInt(that); iGene--;) {
    // If this gene mutes
    if (rnd() < probbMute) {
        hasMuted = true;
        // Get the bounds
        const VecLong2D* const boundsI = GABoundsAdnInt(that, iGene);
        VecFloat2D bounds = VecLongToFloat2D(boundsI);
        // Apply the mutation (as it is int value, ensure the amplitude
        // is big enough to have an effect
        float ampI = MIN(2.0,
            (float)(VecGet(&bounds, 1) - VecGet(&bounds, 0)) * amp);
        GAAdnSetGeneI(child, iGene, GAAdnGetGeneI(child, iGene) +
            (long)round(ampI * (rnd() - 0.5)));
        // Keep the gene value in bounds
        while (GAAdnGetGeneI(child, iGene) < VecGet(&bounds, 0) ||
            GAAdnGetGeneI(child, iGene) > VecGet(&bounds, 1)) {
            if (GAAdnGetGeneI(child, iGene) > VecGet(&bounds, 1))
                GAAdnSetGeneI(child, iGene,
                    2 * VecGet(&bounds, 1) - GAAdnGetGeneI(child, iGene));
            else if (GAAdnGetGeneI(child, iGene) < VecGet(&bounds, 0))
                GAAdnSetGeneI(child, iGene,
                    2 * VecGet(&bounds, 0) - GAAdnGetGeneI(child, iGene));
        }
    }
}
} while (hasMuted == false);
}

// Mute the genes of the entity at rank 'iChild'
// This version is optimised to calculate the parameters of a NeuraNet
// with convolution by muting bases function per cell
void GAMuteNeuraNetConv(GenAlg* const that, const int* const parents,
    const int iChild) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
    if (parents == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'parents' is null");
        PBErrCatch(GenAlgErr);
    }
    if (iChild < 0 || iChild >= GAGetNbAdns(that)) {
        GenAlgErr->_type = PBErrTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<=%d)",
            iChild, GAGetNbAdns(that));
        PBErrCatch(GenAlgErr);
    }

```

```

    }
#endif
    // Get the first parent and child
    GenAlgAdn* parentA = GAAdn(that, parents[0]);
    GenAlgAdn* child = GAAdn(that, iChild);
    // Get the proba amplitude of mutation
    float probMute = sqrt(((float)iChild) / ((float)GAGetNbAdns(that)));
    float amp = 1.0 - sqrt(1.0 / (float)(parentA->_age));
    probMute /= (float)(that->_NNdata._nbLink);
    probMute += (float)(parentA->_age) / 10000.0;
    if (probMute < PBMath_EPSILON)
        probMute = PBMath_EPSILON;
    bool hasMuted = false;
    int nbTry = 0;
    do {
        // For each gene of the adn for floating point value
        for (long iGene = GAGetLengthAdnFloat(that); iGene--;) {
            // If this gene mutes
            if (rnd() < probMute * VecGet(parentA->_mutabilityF, iGene)) {
                hasMuted = true;
                // Get the bounds
                const VecFloat2D* const bounds = GABoundsAdnFloat(that, iGene);
                // Declare a variable to memorize the previous value of the gene
                float prevVal = GAAdnGetGeneF(child, iGene);
                // Apply the mutation
                GAAdnSetGeneF(child, iGene, GAAdnGetGeneF(child, iGene) +
                    (VecGet(bounds, 1) - VecGet(bounds, 0)) * amp *
                    (rnd() - 0.5) + GAAdnGetDeltaGeneF(child, iGene));
                // Keep the gene value in bounds
                while (GAAdnGetGeneF(child, iGene) < VecGet(bounds, 0) ||
                    GAAdnGetGeneF(child, iGene) > VecGet(bounds, 1)) {
                    if (GAAdnGetGeneF(child, iGene) > VecGet(bounds, 1))
                        GAAdnSetGeneF(child, iGene,
                            2.0 * VecGet(bounds, 1) - GAAdnGetGeneF(child, iGene));
                    else if (GAAdnGetGeneF(child, iGene) < VecGet(bounds, 0))
                        GAAdnSetGeneF(child, iGene,
                            2.0 * VecGet(bounds, 0) - GAAdnGetGeneF(child, iGene));
                }
                // Update the deltaAdn
                GAAdnSetDeltaGeneF(child, iGene,
                    GAAdnGetGeneF(child, iGene) - prevVal);
            }
        }
        ++nbTry;
    } while (hasMuted == false && nbTry < 10);
}

// Print the information about the GenAlg 'that' on the stream 'stream'
void GAPrintln(const GenAlg* const that, FILE* const stream) {
    #if BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErrCatch(GenAlgErr);
        }
        if (stream == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'stream' is null");
            PBErrCatch(GenAlgErr);
        }
    #endif
    fprintf(stream, "epoch:%lu\n", GAGetCurEpoch(that));
}

```

```

fprintf(stream, "%d entities, %d elites\n", GAGetNbAdns(that),
        GAGetNbElites(that));
GSetIterBackward iter = GSetIterBackwardCreateStatic(GAAdns(that));
int iEnt = 0;
do {
    GenAlgAdn* ent = GSetIterGet(&iter);
    fprintf(stream, "%d value:%f ", iEnt,
            GSetIterGetElem(&iter)->_sortVal);
    if (iEnt < GAGetNbElites(that))
        fprintf(stream, "elite ");
    GAAdnPrintln(ent, stream);
    ++iEnt;
} while (GSetIterStep(&iter));
}

// Print a summary about the elite entities of the GenAlg 'that'
// on the stream 'stream'
void GAEliteSummaryPrintln(const GenAlg* const that,
        FILE* const stream) {
    #if BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErrCatch(GenAlgErr);
        }
        if (stream == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'stream' is null");
            PBErrCatch(GenAlgErr);
        }
    #endif
    GSetIterBackward iter = GSetIterBackwardCreateStatic(GAAdns(that));
    int iEnt = 0;
    GenAlgAdn* leader = GSetIterGet(&iter);
    fprintf(stream, "(age,val,div) ");
    do {
        GenAlgAdn* ent = GSetIterGet(&iter);
        fprintf(stream, "(%lu,%.3f,%.3f) ", GAAdnGetAge(ent),
                GSetIterGetElem(&iter)->_sortVal,
                GAAdnGetDiversity(ent, leader, that));
        ++iEnt;
    } while (GSetIterStep(&iter) && iEnt < GAGetNbElites(that));
    fprintf(stream, "\n");
}

// Update the norm of the range value for adans of the GenAlg 'that'
void GAUpdateNormRange(GenAlg* const that) {
    #if BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErrCatch(GenAlgErr);
        }
    #endif
    // If there are float adn
    if (GAGetLengthAdnFloat(that) > 0) {
        // Declare a vector to memorize the ranges in float gene values
        VecFloat* range = VecFloatCreate(GAGetLengthAdnFloat(that));
        // Calculate the ranges in gene values
        for (long iGene = GAGetLengthAdnFloat(that); iGene--;)
            VecSet(range, iGene,
                    VecGet(GABoundsAdnFloat(that, iGene), 1) -

```



```

        VecGet(GABoundsAdnFloat(that, iGene), 0));
    // Calculate the norm of the range
    that->_normRangeFloat = VecNorm(range);
    // Free memory
    VecFree(&range);
}

// If there are int adn
if (GAGetLengthAdnInt(that) > 0) {
    // Declare a vector to memorize the ranges in int gene values
    VecFloat* range = VecFloatCreate(GAGetLengthAdnInt(that));
    // Calculate the ranges in gene values
    for (long iGene = GAGetLengthAdnInt(that); iGene--;)
        VecSet(range, iGene,
            VecGet(GABoundsAdnInt(that, iGene), 1) -
            VecGet(GABoundsAdnInt(that, iGene), 0));
    // Calculate the norm of the range
    that->_normRangeInt = VecNorm(range);
    // Free memory
    VecFree(&range);
}
}

// Get the diversity value of 'adnA' against 'adnB'
// The diversity is equal to
float GAAdnGetDiversity(const GenAlgAdn* const adnA,
    const GenAlgAdn* const adnB, const GenAlg* const ga) {
#ifdef BUILDMODE == 0
    if (adnA == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'adnA' is null");
        PBErrCatch(GenAlgErr);
    }
    if (adnB == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'adnB' is null");
        PBErrCatch(GenAlgErr);
    }
}
#endif
// Declare a variable to memorize the result
float diversity = 0.0;
// If there are adn for floating point values
if (GAAdnAdnF(adnA) != NULL && GAAdnAdnF(adnB) != NULL) {
    // Get the difference in adn with the first entity
    VecFloat* diff =
        VecGetOp(GAAdnAdnF(adnA), 1.0, GAAdnAdnF(adnB), -1.0);
    // Calculate the diversity
    diversity += VecNorm(diff) / ga->_normRangeFloat;
    // Free memory
    VecFree(&diff);
}
// If there are adn for int values
if (GAAdnAdnI(adnA) != NULL && GAAdnAdnI(adnB) != NULL) {
    // Get the difference in adn with the first entity
    VecLong* diffI =
        VecGetOp(GAAdnAdnI(adnA), 1, GAAdnAdnI(adnB), -1);
    VecFloat* diff = VecLongToFloat(diffI);
    // Calculate the diversity
    diversity += VecNorm(diff) / ga->_normRangeInt;
    // Free memory
    VecFree(&diffI);
}

```

```

    VecFree(&diff);
}
// Correct diversity if there was both float and int adns
if (GAAdnAdnF(adnA) != NULL && GAAdnAdnF(adnB) != NULL &&
    GAAdnAdnI(adnA) != NULL && GAAdnAdnI(adnB) != NULL)
    diversity /= 2.0;
// Return the result
return diversity;
}

// Function which return the JSON encoding of 'that'
JSONNode* GAAdnEncodeAsJSON(const GenAlgAdn* const that,
    const float elo) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        PBMathErr->_type = PBErrTypeNullPointer;
        sprintf(PBMathErr->_msg, "'that' is null");
        PBErrCatch(PBMathErr);
    }
#endif
    // Create the JSON structure
    JSONNode* json = JSONCreate();
    // Declare a buffer to convert value into string
    char val[100];
    // Encode the id
    sprintf(val, "%lu", that->_id);
    JSONAddProp(json, "_id", val);
    // Encode the age
    sprintf(val, "%lu", that->_age);
    JSONAddProp(json, "_age", val);
    // Encode the elo
    sprintf(val, "%f", elo);
    JSONAddProp(json, "_elo", val);
    // Encode the value
    sprintf(val, "%f", that->_val);
    JSONAddProp(json, "_val", val);
    // Encode the genes
    if (that->_adnF != NULL) {
        JSONAddProp(json, "_adnF", VecEncodeAsJSON(that->_adnF));
        JSONAddProp(json, "_deltaAdnF", VecEncodeAsJSON(that->_deltaAdnF));
    }
    if (that->_adnI != NULL)
        JSONAddProp(json, "_adnI", VecEncodeAsJSON(that->_adnI));
    // Return the created JSON
    return json;
}

// Function which return the JSON encoding of 'that'
JSONNode* GAEncodeAsJSON(const GenAlg* const that) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        PBMathErr->_type = PBErrTypeNullPointer;
        sprintf(PBMathErr->_msg, "'that' is null");
        PBErrCatch(PBMathErr);
    }
#endif
    // Create the JSON structure
    JSONNode* json = JSONCreate();
    // Declare a buffer to convert value into string
    char val[100];
    // Encode the type
    sprintf(val, "%d", GAGetType(that));

```

```

JSONAddProp(json, "_type", val);
switch (GAGetType(that)) {
    case genAlgTypeNeuraNet:
        sprintf(val, "%d", that->_NNdata._nbIn);
        JSONAddProp(json, "NN_nbIn", val);
        sprintf(val, "%d", that->_NNdata._nbHid);
        JSONAddProp(json, "NN_nbHid", val);
        sprintf(val, "%d", that->_NNdata._nbOut);
        JSONAddProp(json, "NN_nbOut", val);
        break;
    case genAlgTypeNeuraNetConv:
        sprintf(val, "%d", that->_NNdata._nbIn);
        JSONAddProp(json, "NN_nbIn", val);
        sprintf(val, "%d", that->_NNdata._nbHid);
        JSONAddProp(json, "NN_nbHid", val);
        sprintf(val, "%d", that->_NNdata._nbOut);
        JSONAddProp(json, "NN_nbOut", val);
        sprintf(val, "%ld", that->_NNdata._nbBaseConv);
        JSONAddProp(json, "NN_nbBaseConv", val);
        sprintf(val, "%ld", that->_NNdata._nbBaseCellConv);
        JSONAddProp(json, "NN_nbBaseCellConv", val);
        sprintf(val, "%ld", that->_NNdata._nbLink);
        JSONAddProp(json, "NN_nbLink", val);
        break;
    default:
        break;
}
// Encode the nb adns
sprintf(val, "%d", GAGetNbAdns(that));
JSONAddProp(json, "_nbAdns", val);
// Encode the nb elites
sprintf(val, "%d", GAGetNbElites(that));
JSONAddProp(json, "_nbElites", val);
// Encode the length adn float
sprintf(val, "%ld", GAGetLengthAdnFloat(that));
JSONAddProp(json, "_lengthAdnF", val);
// Encode the length adn int
sprintf(val, "%ld", GAGetLengthAdnInt(that));
JSONAddProp(json, "_lengthAdnI", val);
// Encode the epoch
sprintf(val, "%lu", GAGetCurEpoch(that));
JSONAddProp(json, "_curEpoch", val);
// Encode the next id
sprintf(val, "%lu", that->_nextId);
JSONAddProp(json, "_nextId", val);
// Encode the bounds
JSONArrayStruct setBoundFloat = JSONArrayStructCreateStatic();
if (GAGetLengthAdnFloat(that) > 0) {
    for (long iBound = 0; iBound < GAGetLengthAdnFloat(that); ++iBound)
        JSONArrayStructAdd(&setBoundFloat,
            VecEncodeAsJSON((VecFloat*)GABoundsAdnFloat(that, iBound)));
    JSONAddProp(json, "_boundFloat", &setBoundFloat);
}
JSONArrayStruct setBoundInt = JSONArrayStructCreateStatic();
if (GAGetLengthAdnInt(that) > 0) {
    for (long iBound = 0; iBound < GAGetLengthAdnInt(that); ++iBound)
        JSONArrayStructAdd(&setBoundInt,
            VecEncodeAsJSON((VecLong*)GABoundsAdnInt(that, iBound)));
    JSONAddProp(json, "_boundInt", &setBoundInt);
}
// Save the adns
JSONArrayStruct setAdn = JSONArrayStructCreateStatic();

```

```

for (int iEnt = 0; iEnt < GAGetNbAdns(that); ++iEnt) {
    GenAlgAdn* ent = GSetElemData(GSetElement(GAAdns(that), iEnt));
    float sortVal = GSetElemGetSortVal(GSetElement(GAAdns(that), iEnt));
    JSONArrayStructAdd(&setAdn, GAAdnEncodeAsJSON(ent, sortVal));
}
JSONAddProp(json, "_adns", &setAdn);
// Save the best adn
JSONAddProp(json, "_bestAdn",
    GAAdnEncodeAsJSON(GABestAdn(that), 0.0));
// Free memory
JSONArrayStructFlush(&setBoundFloat);
JSONArrayStructFlush(&setBoundInt);
JSONArrayStructFlush(&setAdn);
// Return the created JSON
return json;
}

// Function which decode from JSON encoding 'json' to 'that'
bool GAAdnDecodeAsJSON(GenAlgAdn** that, const JSONNode* const json) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        PBMathErr->_type = PBErrTypeNullPointer;
        sprintf(PBMathErr->_msg, "'that' is null");
        PBErrCatch(PBMathErr);
    }
    if (json == NULL) {
        PBMathErr->_type = PBErrTypeNullPointer;
        sprintf(PBMathErr->_msg, "'json' is null");
        PBErrCatch(PBMathErr);
    }
#endif
    // If 'that' is already allocated
    if (*that != NULL)
        // Free memory
        GenAlgAdnFree(that);
    // Get the id from the JSON
    JSONNode* prop = JSONProperty(json, "_id");
    if (prop == NULL) {
        return false;
    }
    unsigned long id = strtoul(JSONLb1Val(prop), NULL, 10);
    // Get the lengthAdnF from the JSON
    long lengthAdnF = 0;
    prop = JSONProperty(json, "_adnF");
    if (prop != NULL) {
        JSONNode* subprop = JSONProperty(prop, "_dim");
        lengthAdnF = atol(JSONLb1Val(subprop));
    }
    // Get the lengthAdnI from the JSON
    long lengthAdnI = 0;
    prop = JSONProperty(json, "_adnI");
    if (prop != NULL) {
        JSONNode* subprop = JSONProperty(prop, "_dim");
        lengthAdnI = atol(JSONLb1Val(subprop));
    }
    // Allocate memory
    *that = GenAlgAdnCreate(id, lengthAdnF, lengthAdnI);
    // Get the age from the JSON
    prop = JSONProperty(json, "_age");
    if (prop == NULL) {
        return false;
    }
}

```

```

(*that)->_age = strtoul(JSONLblVal(prop), NULL, 10);
// Get the adnF from the JSON
prop = JSONProperty(json, "_adnF");
if (prop != NULL) {
    if (!VecDecodeAsJSON(&((*that)->_adnF), prop)) {
        return false;
    }
    prop = JSONProperty(json, "_deltaAdnF");
    if (prop == NULL) {
        return false;
    }
    if (!VecDecodeAsJSON(&((*that)->_deltaAdnF), prop)) {
        return false;
    }
}
// Get the adnI from the JSON
prop = JSONProperty(json, "_adnI");
if (prop != NULL)
    if (!VecDecodeAsJSON(&((*that)->_adnI), prop)) {
        return false;
    }
// Get the value
prop = JSONProperty(json, "_val");
if (prop == NULL) {
    return false;
}
(*that)->_val = atof(JSONLblVal(prop));
// Return the success code
return true;
}

// Function which decode from JSON encoding 'json' to 'that'
bool GADecodeAsJSON(GenAlg** that, const JSONNode* const json) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        PBMathErr->_type = PBErrTypeNullPointer;
        sprintf(PBMathErr->_msg, "'that' is null");
        PBErrCatch(PBMathErr);
    }
    if (json == NULL) {
        PBMathErr->_type = PBErrTypeNullPointer;
        sprintf(PBMathErr->_msg, "'json' is null");
        PBErrCatch(PBMathErr);
    }
#endif
    // If 'that' is already allocated
    if (*that != NULL)
        // Free memory
        GenAlgFree(that);
    // Decode the nb adns
    JSONNode* prop = JSONProperty(json, "_nbAdns");
    if (prop == NULL) {
        return false;
    }
    int nbAdns = atoi(JSONLblVal(prop));
    // Decode the nb elites
    prop = JSONProperty(json, "_nbElites");
    if (prop == NULL) {
        return false;
    }
    int nbElites = atoi(JSONLblVal(prop));
    // Decode the length adn float

```

```

prop = JSONProperty(json, "_lengthAdnF");
if (prop == NULL) {
    return false;
}
long lengthAdnF = atoi(JSONLb1Val(prop));
// Decode the length adn int
prop = JSONProperty(json, "_lengthAdnI");
if (prop == NULL) {
    return false;
}
long lengthAdnI = atoi(JSONLb1Val(prop));
// Allocate memory
*that = GenAlgCreate(nbAdns, nbElites, lengthAdnF, lengthAdnI);
// Decode the type
prop = JSONProperty(json, "_type");
if (prop == NULL) {
    return false;
}
int type = atoi(JSONLb1Val(prop));
int nbIn = 0;
int nbOut = 0;
int nbHid = 0;
switch (type) {
    case genAlgTypeNeuraNet:
        prop = JSONProperty(json, "NN_nbIn");
        if (prop == NULL) {
            return false;
        }
        nbIn = atoi(JSONLb1Val(prop));
        prop = JSONProperty(json, "NN_nbOut");
        if (prop == NULL) {
            return false;
        }
        nbOut = atoi(JSONLb1Val(prop));
        prop = JSONProperty(json, "NN_nbHid");
        if (prop == NULL) {
            return false;
        }
        nbHid = atoi(JSONLb1Val(prop));
        prop = JSONProperty(json, "NN_nbBaseConv");
        if (prop == NULL) {
            return false;
        }
        GASetTypeNeuraNet(*that, nbIn, nbHid, nbOut);
        break;
    case genAlgTypeNeuraNetConv:
        prop = JSONProperty(json, "NN_nbIn");
        if (prop == NULL) {
            return false;
        }
        nbIn = atoi(JSONLb1Val(prop));
        prop = JSONProperty(json, "NN_nbOut");
        if (prop == NULL) {
            return false;
        }
        nbOut = atoi(JSONLb1Val(prop));
        prop = JSONProperty(json, "NN_nbHid");
        if (prop == NULL) {
            return false;
        }
        nbHid = atoi(JSONLb1Val(prop));
        prop = JSONProperty(json, "NN_nbBaseConv");

```

```

        if (prop == NULL) {
            return false;
        }
        long nbBaseConv = atol(JSONLblVal(prop));
        prop = JSONProperty(json, "NN_nbBaseCellConv");
        if (prop == NULL) {
            return false;
        }
        long nbBaseCellConv = atol(JSONLblVal(prop));
        prop = JSONProperty(json, "NN_nbLink");
        if (prop == NULL) {
            return false;
        }
        long nbLink = atol(JSONLblVal(prop));
        GASetTypeNeuraNetConv(*that, nbIn, nbHid, nbOut, nbBaseConv,
            nbBaseCellConv, nbLink);
        break;
    default:
        break;
}
// Decode the epoch
prop = JSONProperty(json, "_curEpoch");
if (prop == NULL) {
    return false;
}
(*that)->_curEpoch =
    strtoul(JSONLblVal(prop), NULL, 10);
// Decode the next id
prop = JSONProperty(json, "_nextId");
if (prop == NULL) {
    return false;
}
(*that)->_nextId = strtoul(JSONLblVal(prop), NULL, 10);
// Decode the bounds
prop = JSONProperty(json, "_boundFloat");
if (prop != NULL) {
    if (JSONGetNbValue(prop) != GAGetLengthAdnFloat(*that))
        return false;
    for (long iBound = 0; iBound < GAGetLengthAdnFloat(*that); ++iBound) {
        JSONNode* val = JSONValue(prop, iBound);
        VecFloat2D* b = NULL;
        if (!VecDecodeAsJSON((VecFloat**) &b, val)) {
            return false;
        }
        GASetBoundsAdnFloat(*that, iBound, b);
        VecFree((VecFloat**) &b);
    }
}
prop = JSONProperty(json, "_boundInt");
if (prop != NULL) {
    if (JSONGetNbValue(prop) != GAGetLengthAdnInt(*that))
        return false;
    for (long iBound = 0; iBound < GAGetLengthAdnInt(*that); ++iBound) {
        JSONNode* val = JSONValue(prop, iBound);
        VecLong2D* b = NULL;
        if (!VecDecodeAsJSON((VecLong**) &b, val)) {
            return false;
        }
        GASetBoundsAdnInt(*that, iBound, b);
        VecFree((VecLong**) &b);
    }
}
}

```

```

// Upadte the norm of the range values
GAUpdateNormRange(*that);
// Decode the adns
prop = JSONProperty(json, "_adns");
if (prop == NULL) {
    return false;
}
if (JSONGetNbValue(prop) != GAGetNbAdns(*that))
    return false;
for (int iEnt = 0; iEnt < GAGetNbAdns(*that); ++iEnt) {
    JSONNode* val = JSONValue(prop, iEnt);
    if (!GAAdnDecodeAsJSON(
        (GenAlgAdn*)&(GSetElement(GAAAdns(*that), iEnt)->_data), val)) {
        return false;
    }
}
// Decode the best adn
prop = JSONProperty(json, "_bestAdn");
if (prop == NULL) {
    return false;
}
if (!GAAdnDecodeAsJSON((GenAlgAdn*)&((*that)->_bestAdn), prop)) {
    return false;
}

// Return the success code
return true;
}

// Load the GenAlg 'that' from the stream 'stream'
// If the GenAlg is already allocated, it is freed before loading
// Return true in case of success, else false
bool GALoad(GenAlg** that, FILE* const stream) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
    if (stream == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'stream' is null");
        PBErrCatch(GenAlgErr);
    }
}
#endif
// Declare a json to load the encoded data
JSONNode* json = JSONCreate();
// Load the whole encoded data
if (!JSONLoad(json, stream)) {
    return false;
}
// Decode the data from the JSON
if (!GADecodeAsJSON(that, json)) {
    return false;
}
// Free the memory used by the JSON
JSONFree(&json);
// Return the success code
return true;
}

// Save the GenAlg 'that' to the stream 'stream'

```



```

// If 'compact' equals true it saves in compact form, else it saves in
// readable form
// Return true in case of success, else false
bool GASave(const GenAlg* const that, FILE* const stream,
            const bool compact) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
    if (stream == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'stream' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    // Get the JSON encoding
    JSONNode* json = GAEncodeAsJSON(that);
    // Save the JSON
    if (!JSONSave(json, stream, compact)) {
        return false;
    }
    // Free memory
    JSONFree(&jjson);
    // Return success code
    return true;
}

// Set the flag memorizing if the TextOMeter is displayed for
// the GenAlg 'that' to 'flag'
void GASetTextOMeterFlag(GenAlg* const that, bool flag) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    // If the requested flag is different from the current flag;
    if (that->_flagTextOMeter != flag) {
        if (flag && that->_textOMeter == NULL) {
            char title[] = "GenAlg";
            int width = strlen(GENALG_TXTOMETER_LINE1) + 1;
            int height = 10 +
                MIN(GENALG_TXTOMETER_NBADNDISPLAYED, GAGetNbMaxAdn(that));
            that->_textOMeter = TextOMeterCreate(title, width, height);
        }
        if (!flag && that->_textOMeter != NULL) {
            TextOMeterFree(&(that->_textOMeter));
        }
        that->_flagTextOMeter = flag;
    }
}

// Refresh the content of the TextOMeter attached to the GenAlg 'that'
void GAUpdateTextOMeter(const GenAlg* const that) {
#ifdef BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
}

```

```

}
if (that->_textOMeter == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that->_textOMeter' is null");
    PBErrCatch(GenAlgErr);
}
#endif
// Clear the TextOMeter
TextOMeterClear(that->_textOMeter);
// Declare a variable to print the content of the TextOMeter
char str[50];
// Print the content of the TextOMeter
// Epoch #xxxxxx KTEvent #xxxxxx
sprintf(str, GENALG_TXTOMETER_FORMAT1,
    GAGetCurEpoch(that), GAGetNbKTEvent(that));
TextOMeterPrint(that->_textOMeter, str);
// Diversity +xxxxxx.xxxxxx
sprintf(str, GENALG_TXTOMETER_FORMAT5, GAGetDiversity(that),
    GAGetDiversityThreshold(that));
TextOMeterPrint(that->_textOMeter, str);
// Nb adns xxxxxx
sprintf(str, GENALG_TXTOMETER_FORMAT6, GAGetNbAdns(that));
TextOMeterPrint(that->_textOMeter, str);
//
sprintf(str, "\n");
TextOMeterPrint(that->_textOMeter, str);
// Id      Age      Val
sprintf(str, GENALG_TXTOMETER_LINE2);
TextOMeterPrint(that->_textOMeter, str);
// xxxxxx xxxxxx +xxxxxx.xxxx
sprintf(str, GENALG_TXTOMETER_FORMAT3,
    GAAdnGetId(GABestAdn(that)), GAAdnGetAge(GABestAdn(that)),
    GAAdnGetVal(GABestAdn(that)));
TextOMeterPrint(that->_textOMeter, str);
// .....
sprintf(str, GENALG_TXTOMETER_LINE4);
TextOMeterPrint(that->_textOMeter, str);
// xxxxxx xxxxxx +xxxxxx.xxxx
for (int iRank = 0; iRank < GAGetNbElites(that); ++iRank) {
    sprintf(str, GENALG_TXTOMETER_FORMAT3,
        GAAdnGetId(GAAdn(that, iRank)), GAAdnGetAge(GAAdn(that, iRank)),
        GAAdnGetVal(GAAdn(that, iRank)));
    TextOMeterPrint(that->_textOMeter, str);
}
// .....
sprintf(str, GENALG_TXTOMETER_LINE4);
TextOMeterPrint(that->_textOMeter, str);
// xxxxxx xxxxxx +xxxxxx.xxxx
int maxRank = MIN(GENALG_TXTOMETER_NBADNDISPLAYED, GAGetNbAdns(that));
for (int iRank = GAGetNbElites(that); iRank < maxRank; ++iRank) {
    sprintf(str, GENALG_TXTOMETER_FORMAT3,
        GAAdnGetId(GAAdn(that, iRank)), GAAdnGetAge(GAAdn(that, iRank)),
        GAAdnGetVal(GAAdn(that, iRank)));
    TextOMeterPrint(that->_textOMeter, str);
}
// Fill in with blank lines if necessary
sprintf(str, "\n");
for (int iBlank = GAGetNbAdns(that);
    iBlank < GENALG_TXTOMETER_NBADNDISPLAYED; ++iBlank) {
    TextOMeterPrint(that->_textOMeter, str);
}
// If there are more adns than available space in the TextOMeter

```

```

    if (GAGetNbAdns(that)> GENALG_TXTOMETER_NBADNDISPLAYED) {
        sprintf(str, "...");
        TextOMeterPrint(that->_textOMeter, str);
    }
    // Flush the content of the TextOMeter
    TextOMeterFlush(that->_textOMeter);
}

```

## 3.2 genalg-inline.c

```

// ===== GENALG-static inline.C =====

// ----- GenAlgAdn

// ===== Functions implementation =====

// Return the adn for floating point values of the GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
const VecFloat* GAAdnAdnF(const GenAlgAdn* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    return that->_adnF;
}

// Return the delta of adn for floating point values of the
// GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
const VecFloat* GAAdnDeltaAdnF(const GenAlgAdn* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    return that->_deltaAdnF;
}

// Return the adn for integer values of the GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
VecLong* GAAdnAdnI(const GenAlgAdn* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
}

```

```

    return that->_adnI;
}

// Get the 'iGene'-th gene of the adn for floating point values of the
// GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
float GAAdnGetGeneF(const GenAlgAdn* const that, const long iGene) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    return VecGet(that->_adnF, iGene);
}

// Get the delta of the 'iGene'-th gene of the adn for floating point
// values of the GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
float GAAdnGetDeltaGeneF(const GenAlgAdn* const that, const long iGene) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    return VecGet(that->_deltaAdnF, iGene);
}

// Get the 'iGene'-th gene of the adn for int values of the
// GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
int GAAdnGetGeneI(const GenAlgAdn* const that, const long iGene) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    return VecGet(that->_adnI, iGene);
}

// Set the 'iGene'-th gene of the adn for floating point values of the
// GenAlgAdn 'that' to 'gene'
#if BUILDMODE != 0
static inline
#endif
void GAAdnSetGeneF(GenAlgAdn* const that, const long iGene,
    const float gene) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
    }
#endif
}

```

```

        PBErCatch(GenAlgErr);
    }
#endif
    VecSet(that->_adnF, iGene, gene);
}

// Set the delta of the 'iGene'-th gene of the adn for floating point
// values of the GenAlgAdn 'that' to 'delta'
#if BUILDMODE != 0
static inline
#endif
void GAAdnSetDeltaGeneF(GenAlgAdn* const that, const long iGene,
    const float delta) {
    #if BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErCatch(GenAlgErr);
        }
    #endif
    VecSet(that->_deltaAdnF, iGene, delta);
}

// Set the 'iGene'-th gene of the adn for int values of the
// GenAlgAdn 'that' to 'gene'
#if BUILDMODE != 0
static inline
#endif
void GAAdnSetGeneI(GenAlgAdn* const that, const long iGene,
    const long gene) {
    #if BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErCatch(GenAlgErr);
        }
    #endif
    VecSet(that->_adnI, iGene, gene);
}

// Get the id of the GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned long int GAAdnGetId(const GenAlgAdn* const that) {
    #if BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErCatch(GenAlgErr);
        }
    #endif
    return that->_id;
}

// Get the age of the GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned long int GAAdnGetAge(const GenAlgAdn* const that) {
    #if BUILDMODE == 0
        if (that == NULL) {

```

```

        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    return that->_age;
}

// Get the value of the GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
float GAAdnGetVal(const GenAlgAdn* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    return that->_val;
}

// Return true if the GenAlgAdn 'that' is new, i.e. is age equals 1
// Return false
#if BUILDMODE != 0
static inline
#endif
bool GAAdnIsNew(const GenAlgAdn* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    return (that->_age == 1);
}

// Copy the GenAlgAdn 'tho' into the GenAlgAdn 'that'
#if BUILDMODE != 0
static inline
#endif
void GAAdnCopy(GenAlgAdn* const that, const GenAlgAdn* const tho) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
    if (tho == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'tho' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    that->_id = tho->_id;
    that->_age = tho->_age;
    that->_val = tho->_val;
    if (tho->_adnF != NULL)
        VecCopy(that->_adnF, tho->_adnF);
    else

```

```

    VecFree(&(that->_adnF));
    if (tho->_deltaAdnF != NULL)
        VecCopy(that->_deltaAdnF, tho->_deltaAdnF);
    else
        VecFree(&(that->_deltaAdnF));
    if (tho->_adnI != NULL)
        VecCopy(that->_adnI, tho->_adnI);
    else
        VecFree(&(that->_adnI));
}

// Set the mutability vectors for the GenAlgAdn 'that' to 'mutability'
#if BUILDMODE != 0
static inline
#endif
void GAAdnSetMutabilityInt(GenAlgAdn* const that,
    const VecFloat* const mutability) {
    #if BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErrCatch(GenAlgErr);
        }
        if (that->_mutabilityI == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that->_mutabilityI' is null");
            PBErrCatch(GenAlgErr);
        }
        if (mutability == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'mutability' is null");
            PBErrCatch(GenAlgErr);
        }
        if (VecGetDim(mutability) != VecGetDim(GAAdnAdnF(that))) {
            GenAlgErr->_type = PBErrTypeInvalidArg;
            sprintf(GenAlgErr->_msg, "'mutability's dim is invalid (%ld==%ld)",
                VecGetDim(mutability), VecGetDim(GAAdnAdnI(that)));
            PBErrCatch(GenAlgErr);
        }
    #endif
    VecCopy(that->_mutabilityI, mutability);
}

#if BUILDMODE != 0
static inline
#endif
void GAAdnSetMutabilityFloat(GenAlgAdn* const that,
    const VecFloat* const mutability) {
    #if BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErrCatch(GenAlgErr);
        }
        if (that->_mutabilityF == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that->_mutabilityF' is null");
            PBErrCatch(GenAlgErr);
        }
        if (mutability == NULL) {
            GenAlgErr->_type = PBErrTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'mutability' is null");
        }
    #endif
}

```

```

        PBErCatch(GenAlgErr);
    }
    if (VecGetDim(mutability) != VecGetDim(GAAdnAdnF(that))) {
        GenAlgErr->_type = PBErTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "'mutability's dim is invalid (%ld==%ld)",
            VecGetDim(mutability), VecGetDim(GAAdnAdnF(that)));
        PBErCatch(GenAlgErr);
    }
#endif
    VecCopy(that->_mutabilityF, mutability);
}

// ----- GenAlg

// ===== Functions implementation =====

// Get the type of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
GenAlgType GAGGetType(const GenAlg* const that) {
    #if BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErCatch(GenAlgErr);
        }
    #endif
    return that->_type;
}

// Set the type of the GenAlg 'that' to genAlgTypeNeuraNet, the GenAlg
// will be used with a NeuraNet having 'nbIn' inputs, 'nbHid' hidden
// values and 'nbOut' outputs
#if BUILDMODE != 0
static inline
#endif
void GASetTypeNeuraNet(GenAlg* const that, const int nbIn,
    const int nbHid, const int nbOut) {
    #if BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErCatch(GenAlgErr);
        }
    #endif
    if (GAGetLengthAdnFloat(that) != GAGetLengthAdnInt(that)) {
        GenAlgErr->_type = PBErTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "Must have the same nb of bases and links");
        PBErCatch(GenAlgErr);
    }
    that->_type = genAlgTypeNeuraNet;
    that->_NNdata._nbIn = nbIn;
    that->_NNdata._nbHid = nbHid;
    that->_NNdata._nbOut = nbOut;
    that->_NNdata._nbBaseConv = 0;
}

// Set the type of the GenAlg 'that' to genAlgTypeNeuraNetConv,
// the GenAlg will be used with a NeuraNet having 'nbIn' inputs,
// 'nbHid' hidden values, 'nbOut' outputs, 'nbBaseConv' bases function,
// 'nbLink' links dedicated to the convolution and 'nbBaseCellConv' bases function per cell of convolution

```



```

#if BUILDMODE != 0
static inline
#endif
void GSetTypeNeuraNetConv(GenAlg* const that, const int nbIn,
    const int nbHid, const int nbOut, const long nbBaseConv,
    const long nbBaseCellConv, const long nbLink) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PErrCatch(GenAlgErr);
    }
#endif
    that->_type = genAlgTypeNeuraNetConv;
    that->_NNdata._nbIn = nbIn;
    that->_NNdata._nbHid = nbHid;
    that->_NNdata._nbOut = nbOut;
    that->_NNdata._nbBaseConv = nbBaseConv;
    that->_NNdata._nbBaseCellConv = nbBaseCellConv;
    that->_NNdata._nbLink = nbLink;
}

// Return the GSet of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
GSet* GAAdns(const GenAlg* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PErrCatch(GenAlgErr);
    }
#endif
    return that->_adns;
}

// Return the nb of entities of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
int GAGetNbAdns(const GenAlg* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PErrCatch(GenAlgErr);
    }
#endif
    return GSetNbElem(that->_adns);
}

// Return the nb of elites of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
int GAGetNbElites(const GenAlg* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PErrCatch(GenAlgErr);
    }
#endif
}

```

```

    }
#endif
    return that->_nbElites;
}

// Return the current epoch of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned long int GAGetCurEpoch(const GenAlg* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PErrCatch(GenAlgErr);
    }
#endif
    return that->_curEpoch;
}

// Return the number of KTEvent of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
unsigned long int GAGetNbKTEvent(const GenAlg* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PErrCatch(GenAlgErr);
    }
#endif
    return that->_nbKTEvent;
}

// Return the min nb of adns of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
int GAGetNbMinAdn(const GenAlg* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PErrCatch(GenAlgErr);
    }
#endif
    return that->_nbMinAdn;
}

// Return the max nb of adns of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
int GAGetNbMaxAdn(const GenAlg* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PErrCatch(GenAlgErr);
    }
#endif
}

```

```

    return that->_nbMaxAdn;
}

// Set the min nb of adns of the GenAlg 'that' to 'nb'
#if BUILDMODE != 0
static inline
#endif
void GASetNbMaxAdn(GenAlg* const that, const int nb) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PErrCatch(GenAlgErr);
    }
#endif
    that->_nbMaxAdn = MAX(nb, GAGetNbElites(that) + 1);
    if (GAGetNbMinAdn(that) > that->_nbMaxAdn)
        GASetNbMinAdn(that, nb);
}

// Set the min nb of adns of the GenAlg 'that' to 'nb'
#if BUILDMODE != 0
static inline
#endif
void GASetNbMinAdn(GenAlg* const that, const int nb) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PErrCatch(GenAlgErr);
    }
#endif
    that->_nbMinAdn = MAX(nb, GAGetNbElites(that) + 1);
    if (GAGetNbMaxAdn(that) < that->_nbMinAdn)
        GASetNbMaxAdn(that, nb);
}

// Get the length of adn for floating point value
#if BUILDMODE != 0
static inline
#endif
long GAGetLengthAdnFloat(const GenAlg* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PErrCatch(GenAlgErr);
    }
#endif
    return that->_lengthAdnF;
}

// Get the length of adn for integer value
#if BUILDMODE != 0
static inline
#endif
long GAGetLengthAdnInt(const GenAlg* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PErrCatch(GenAlgErr);
    }
#endif
}

```

```

    }
#endif
    return that->_lengthAdnI;
}

// Set the bounds for the 'iGene'-th gene of adn for floating point
// values to a copy of 'bounds'
#if BUILDMODE != 0
static inline
#endif
void GASetBoundsAdnFloat(GenAlg* const that, const long iGene,
    const VecFloat2D* const bounds) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
    if (bounds == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'bounds' is null");
        PBErrCatch(GenAlgErr);
    }
    if (VecGet(bounds, 0) >= VecGet(bounds, 1)) {
        GenAlgErr->_type = PBErrTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "'bounds' is invalid (%f<%f)",
            VecGet(bounds, 0), VecGet(bounds, 1));
        PBErrCatch(GenAlgErr);
    }
    if (iGene < 0 || iGene >= that->_lengthAdnF) {
        GenAlgErr->_type = PBErrTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "'iGene' is invalid (0<=%ld<=%ld)",
            iGene, that->_lengthAdnF);
        PBErrCatch(GenAlgErr);
    }
#endif
    VecCopy(that->_boundsF + iGene, bounds);
    GAUpdateNormRange(that);
}

// Set the bounds for the 'iGene'-th gene of adn for integer values
// to a copy of 'bounds'
#if BUILDMODE != 0
static inline
#endif
void GASetBoundsAdnInt(GenAlg* const that, const long iGene,
    const VecLong2D* const bounds) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
    if (bounds == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'bounds' is null");
        PBErrCatch(GenAlgErr);
    }
    if (VecGet(bounds, 0) >= VecGet(bounds, 1)) {
        GenAlgErr->_type = PBErrTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "'bounds' is invalid (%ld<%ld)",
            VecGet(bounds, 0), VecGet(bounds, 1));
    }
#endif
}

```

```

        PBErCatch(GenAlgErr);
    }
    if (iGene < 0 || iGene >= that->_lengthAdnI) {
        GenAlgErr->_type = PBErTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "'iGene' is invalid (0<=%ld<%ld)",
            iGene, that->_lengthAdnI);
        PBErCatch(GenAlgErr);
    }
#endif
    VecCopy(that->_boundsI + iGene, bounds);
    GAUpdateNormRange(that);
}

// Get the bounds for the 'iGene'-th gene of adn for floating point
// values
#if BUILDMODE != 0
static inline
#endif
const VecFloat2D* GABoundsAdnFloat(const GenAlg* const that,
    const long iGene) {
    if (BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErCatch(GenAlgErr);
        }
        if (iGene < 0 || iGene >= that->_lengthAdnF) {
            GenAlgErr->_type = PBErTypeInvalidArg;
            sprintf(GenAlgErr->_msg, "'iGene' is invalid (0<=%ld<%ld)",
                iGene, that->_lengthAdnF);
            PBErCatch(GenAlgErr);
        }
    )
    #endif
    return that->_boundsF + iGene;
}

// Get the bounds for the 'iGene'-th gene of adn for integer values
#if BUILDMODE != 0
static inline
#endif
const VecLong2D* GABoundsAdnInt(const GenAlg* const that,
    const long iGene) {
    if (BUILDMODE == 0
        if (that == NULL) {
            GenAlgErr->_type = PBErTypeNullPointer;
            sprintf(GenAlgErr->_msg, "'that' is null");
            PBErCatch(GenAlgErr);
        }
        if (iGene < 0 || iGene >= that->_lengthAdnI) {
            GenAlgErr->_type = PBErTypeInvalidArg;
            sprintf(GenAlgErr->_msg, "'iGene' is invalid (0<=%ld<%ld)",
                iGene, that->_lengthAdnI);
            PBErCatch(GenAlgErr);
        }
    )
    #endif
    return that->_boundsI + iGene;
}

// Get the GenAlgAdn of the GenAlg 'that' currently at rank 'iRank'
// (0 is the best adn)
#if BUILDMODE != 0
static inline

```

```

#endif
GenAlgAdn* GAAdn(const GenAlg* const that, const int iRank) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
    if (iRank < -1 || iRank >= GAGetNbAdns(that)) {
        GenAlgErr->_type = PBErrTypeInvalidArg;
        sprintf(GenAlgErr->_msg, "'iRank' is invalid (0<=%d<%d)",
            iRank, GAGetNbAdns(that));
        PBErrCatch(GenAlgErr);
    }
#endif
    if (iRank == -1)
        return (GenAlgAdn*)GABestAdn(that);
    else
        return (GenAlgAdn*)GSetGet(that->_adns,
            GSetNbElem(that->_adns) - iRank - 1);
}

// Set the value of the GenAlgAdn 'adn' of the GenAlg 'that' to 'val'
#if BUILDMODE != 0
static inline
#endif
void GASetAdnValue(GenAlg* const that, GenAlgAdn* const adn,
    const float val) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
    if (adn == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'adn' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    // Set the value
    adn->_val = val;
    GSetElemSetSortVal((GSetElem*)GSetFirstElem(GAAdns(that), adn), val);
}

// Get the diversity of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
float GAGetDiversity(const GenAlg* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    float diversity = GAGetDiversityThreshold(that) + 1.0;
    for (int iAdn = 0; iAdn < GAGetNbElites(that) - 1; ++iAdn) {
        for (int jAdn = iAdn + 1; jAdn < GAGetNbElites(that); ++jAdn) {
            diversity = fabs(MIN(diversity,
                GAAdn(that, iAdn)->_val - GAAdn(that, jAdn)->_val));
        }
    }
}

```

```

    }
}
return diversity;
}

// Get the diversity threshold of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
float GAGetDiversityThreshold(const GenAlg* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    return that->_diversityThreshold;
}

// Set the diversity threshold of the GenAlg 'that' to 'threshold'
#if BUILDMODE != 0
static inline
#endif
void GASetDiversityThreshold(GenAlg* const that, const float threshold) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    that->_diversityThreshold = threshold;
}

// Return the best adn of the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
const GenAlgAdn* GABestAdn(const GenAlg* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
#endif
    return that->_bestAdn;
}

// Return the flag memorizing if the TextOMeter is displayed for
// the GenAlg 'that'
#if BUILDMODE != 0
static inline
#endif
bool GAIstextOMeterActive(const GenAlg* const that) {
#if BUILDMODE == 0
    if (that == NULL) {
        GenAlgErr->_type = PBErrTypeNullPointer;
        sprintf(GenAlgErr->_msg, "'that' is null");
        PBErrCatch(GenAlgErr);
    }
}

```

```

#endif
    return that->_flagTextOMeter;
}

```

## 4 Makefile

```

# Build mode
# 0: development (max safety, no optimisation)
# 1: release (min safety, optimisation)
# 2: fast and furious (no safety, optimisation)
BUILD_MODE?=1

all: pbmake_wget main

# Automatic installation of the repository PBMake in the parent folder
pbmake_wget:
if [ ! -d ../PBMake ]; then wget https://github.com/BayashiPascal/PBMake/archive/master.zip; unzip master.zip; rm -f

# Makefile definitions
MAKEFILE_INC=../PBMake/Makefile.inc
include $(MAKEFILE_INC)

# Rules to make the executable
repo=genalg
$$(repo)_EXENAME: \
$$(repo)_EXENAME.o \
$$(repo)_EXE_DEP \
$$(repo)_DEP
$(COMPILER) 'echo "$$(repo)_EXE_DEP $$(repo)_EXENAME.o" | tr ' ' '\n' | sort -u' $(LINK_ARG) $$(repo)_LINK_ARG)

$$(repo)_EXENAME.o: \
$$(repo)_DIR)/$$(repo)_EXENAME.c \
$$(repo)_INC_H_EXE \
$$(repo)_EXE_DEP
$(COMPILER) $(BUILD_ARG) $$(repo)_BUILD_ARG 'echo "$$(repo)_INC_DIR" | tr ' ' '\n' | sort -u' -c $$(repo)_DIR)/

```

## 5 Unit tests

```

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>
#include "genalg.h"

#define RANDOMSEED 2

void UnitTestGenAlgAdnCreateFree() {
    unsigned long int id = 1;
    int lengthAdnF = 2;
    int lengthAdnI = 3;
    GenAlgAdn* ent = GenAlgAdnCreate(id, lengthAdnF, lengthAdnI);
    if (ent->_age != 1 ||

```



```

    ent->_id != id ||
    VecGetDim(ent->_adnF) != lengthAdnF ||
    VecGetDim(ent->_deltaAdnF) != lengthAdnF ||
    VecGetDim(ent->_adnI) != lengthAdnI) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GenAlgAdnCreate failed");
    PBErrCatch(GenAlgErr);
}
GenAlgAdnFree(&ent);
if (ent != NULL) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GenAlgAdnFree failed");
    PBErrCatch(GenAlgErr);
}
printf("UnitTestGenAlgAdnCreateFree OK\n");
}

void UnitTestGenAlgAdnGetSet() {
    unsigned long int id = 1;
    int lengthAdnF = 2;
    int lengthAdnI = 3;
    GenAlgAdn* ent = GenAlgAdnCreate(id, lengthAdnF, lengthAdnI);
    if (GAAdnAdnF(ent) != ent->_adnF) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAAdnAdnF failed");
        PBErrCatch(GenAlgErr);
    }
    if (GAAdnDeltaAdnF(ent) != ent->_deltaAdnF) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAAdnDeltaAdnF failed");
        PBErrCatch(GenAlgErr);
    }
    if (GAAdnAdnI(ent) != ent->_adnI) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAAdnAdnI failed");
        PBErrCatch(GenAlgErr);
    }
    GAAdnSetGeneF(ent, 0, 1.0);
    if (ISEQUALF(VecGet(ent->_adnF, 0), 1.0) == false) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAAdnSetGeneF failed");
        PBErrCatch(GenAlgErr);
    }
    if (ISEQUALF(GAAdnGetGeneF(ent, 0), 1.0) == false) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAAdnGetGeneF failed");
        PBErrCatch(GenAlgErr);
    }
    GAAdnSetDeltaGeneF(ent, 0, 2.0);
    if (ISEQUALF(VecGet(ent->_deltaAdnF, 0), 2.0) == false) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAAdnSetDeltaGeneF failed");
        PBErrCatch(GenAlgErr);
    }
    if (ISEQUALF(GAAdnGetDeltaGeneF(ent, 0), 2.0) == false) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAAdnGetDeltaGeneF failed");
        PBErrCatch(GenAlgErr);
    }
    GAAdnSetGeneI(ent, 0, 3);
    if (VecGet(ent->_adnI, 0) != 3) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
    }
}

```

```

        sprintf(GenAlgErr->_msg, "GAAdnSetGeneI failed");
        PBErrCatch(GenAlgErr);
    }
    if (GAAdnGetGeneI(ent, 0) != 3) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAAdnGetGeneI failed");
        PBErrCatch(GenAlgErr);
    }
    if (GAAdnGetAge(ent) != 1) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAAdnGetAge failed");
        PBErrCatch(GenAlgErr);
    }
    ent->_val = 2.0;
    if (ISEQUALF(GAAdnGetVal(ent), 2.0) == false) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAAdnGetVal failed");
        PBErrCatch(GenAlgErr);
    }
    if (GAAdnGetId(ent) != id) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAAdnGetId failed");
        PBErrCatch(GenAlgErr);
    }
    if (GAAdnIsNew(ent) != true) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAAdnIsNew failed");
        PBErrCatch(GenAlgErr);
    }
    ent->_age = 2;
    if (GAAdnIsNew(ent) != false) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAAdnIsNew failed");
        PBErrCatch(GenAlgErr);
    }
    GenAlgAdnFree(&ent);
    printf("UnitTestGenAlgAdnGetSet OK\n");
}

void UnitTestGenAlgAdnInit() {
    srand(5);
    unsigned long int id = 1;
    int lengthAdnF = 2;
    int lengthAdnI = 2;
    GenAlgAdn* ent = GenAlgAdnCreate(id, lengthAdnF, lengthAdnI);
    GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
        lengthAdnF, lengthAdnI);
    VecFloat2D boundsF = VecFloatCreateStatic2D();
    VecLong2D boundsI = VecLongCreateStatic2D();
    VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
    VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
    GASetBoundsAdnFloat(ga, 0, &boundsF);
    GASetBoundsAdnFloat(ga, 1, &boundsF);
    GASetBoundsAdnInt(ga, 0, &boundsI);
    GASetBoundsAdnInt(ga, 1, &boundsI);
    GAAdnInit(ent, ga);
    if (ISEQUALF(VecGet(ent->_adnF, 0), -0.907064) == false ||
        ISEQUALF(VecGet(ent->_adnF, 1), -0.450509) == false ||
        VecGet(ent->_adnI, 0) != 2 ||
        VecGet(ent->_adnI, 1) != 10) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAAdnInit failed");
    }
}

```

```

        PBErrCatch(GenAlgErr);
    }
    GenAlgFree(&ga);
    GenAlgAdnFree(&ent);
    printf("UnitTestGenAlgAdnInit OK\n");
}

void UnitTestGenAlgAdn() {
    UnitTestGenAlgAdnCreateFree();
    UnitTestGenAlgAdnGetSet();
    UnitTestGenAlgAdnInit();
    printf("UnitTestGenAlgAdn OK\n");
}

void UnitTestGenAlgCreateFree() {
    int lengthAdnF = 2;
    int lengthAdnI = 3;
    GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
        lengthAdnF, lengthAdnI);
    if (ga->_type != genAlgTypeDefault ||
        ga->_curEpoch != 0 ||
        ga->_nbKTEvent != 0 ||
        ga->_nextId != GENALG_NBENTITIES ||
        ga->_nbElites != GENALG_NBELITES ||
        ga->_lengthAdnF != lengthAdnF ||
        ga->_lengthAdnI != lengthAdnI ||
        ga->_flagTextOMeter != false ||
        ga->_nbMinAdn != GENALG_NBENTITIES ||
        ga->_nbMaxAdn != GENALG_NBENTITIES ||
        ISEQUALF(ga->_diversityThreshold, 0.01) != true ||
        ga->_textOMeter != NULL ||
        GSetNbElem(GAAdns(ga)) != GENALG_NBENTITIES) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GenAlgCreate failed");
        PBErrCatch(GenAlgErr);
    }
    GenAlgFree(&ga);
    if (ga != NULL) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GenAlgFree failed");
        PBErrCatch(GenAlgErr);
    }
    printf("UnitTestGenAlgCreateFree OK\n");
}

void UnitTestGenAlgGetSet() {
    int lengthAdnF = 2;
    int lengthAdnI = 3;
    GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
        lengthAdnF, lengthAdnI);
    if (GAGetType(ga) != ga->_type) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAGetType failed");
        PBErrCatch(GenAlgErr);
    }
    if (GAAdns(ga) != ga->_adns) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAElorank failed");
        PBErrCatch(GenAlgErr);
    }
    if (GAGetNbAdns(ga) != GENALG_NBENTITIES) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
    }
}

```

```

    sprintf(GenAlgErr->_msg, "GAGetNbAdns failed");
    PBErrCatch(GenAlgErr);
}
if (GAGetNbElites(ga) != GENALG_NBELITES) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetNbElites failed");
    PBErrCatch(GenAlgErr);
}
if (GAGetCurEpoch(ga) != 0) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetCurEpoch failed");
    PBErrCatch(GenAlgErr);
}
if (GAGetNbKTEvent(ga) != 0) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetNbKTEvent failed");
    PBErrCatch(GenAlgErr);
}
if (ISEQUALF(GAGetDiversityThreshold(ga),
    ga->_diversityThreshold) != true) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetDiversityThreshold failed");
    PBErrCatch(GenAlgErr);
}
GASetDiversityThreshold(ga, 2.0);
if (ISEQUALF(GAGetDiversityThreshold(ga), 2.0) != true) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GASetDiversityThrehsold failed");
    PBErrCatch(GenAlgErr);
}
GASetNbEntities(ga, 10);
if (GAGetNbAdns(ga) != 10 ||
    GAGetNbElites(ga) != 9 ||
    GSetNbElem(GAAdns(ga)) != 10) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GASetNbEntities failed");
    PBErrCatch(GenAlgErr);
}
GASetNbElites(ga, 20);
if (GAGetNbAdns(ga) != 21 ||
    GAGetNbElites(ga) != 20 ||
    GSetNbElem(GAAdns(ga)) != 21) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GASetNbElites failed");
    PBErrCatch(GenAlgErr);
}
if (GAGetLengthAdnFloat(ga) != lengthAdnF) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetLengthAdnFloat failed");
    PBErrCatch(GenAlgErr);
}
if (GAGetLengthAdnInt(ga) != lengthAdnI) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetLengthAdnInt failed");
    PBErrCatch(GenAlgErr);
}
if (GABoundsAdnFloat(ga, 1) != ga->_boundsF + 1) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GABoundsAdnFloat failed");
    PBErrCatch(GenAlgErr);
}
VecFloat2D boundsF = VecFloatCreateStatic2D();

```

```

VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
GASetBoundsAdnFloat(ga, 1, &boundsF);
if (VecIsEqual(GABoundsAdnFloat(ga, 1), &boundsF) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GASetBoundsAdnFloat failed");
    PBErrCatch(GenAlgErr);
}
VecLong2D boundsS = VecLongCreateStatic2D();
VecSet(&boundsS, 0, -1); VecSet(&boundsS, 1, 1);
GASetBoundsAdnInt(ga, 1, &boundsS);
if (VecIsEqual(GABoundsAdnInt(ga, 1), &boundsS) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GASetBoundsAdnInt failed");
    PBErrCatch(GenAlgErr);
}
if (GABoundsAdnInt(ga, 1) != ga->_boundsI + 1) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GABoundsAdnInt failed");
    PBErrCatch(GenAlgErr);
}
GASetAdnValue(ga, GAAdn(ga, 0), 1.0);
if (ISEQUALF(GAAdn(ga, 0)->_val, 1.0) == false ||
    ISEQUALF(ga->_adns->_tail->_sortVal, 1.0) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GASetAdnValue failed");
    PBErrCatch(GenAlgErr);
}
if (GAGetNbMaxAdn(ga) != ga->_nbMaxAdn) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetNbMaxAdn failed");
    PBErrCatch(GenAlgErr);
}
if (GAGetNbMinAdn(ga) != ga->_nbMinAdn) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetNbMinAdn failed");
    PBErrCatch(GenAlgErr);
}
GASetNbMaxAdn(ga, 100);
if (GAGetNbMaxAdn(ga) != 100) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GASetNbMaxAdn failed");
    PBErrCatch(GenAlgErr);
}
GASetNbMinAdn(ga, 100);
if (GAGetNbMinAdn(ga) != 100) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GASetNbMinAdn failed");
    PBErrCatch(GenAlgErr);
}
GenAlgFree(&ga);
ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES, 3, 3);
GASetTypeNeuraNet(ga, 1, 2, 3);
if (GAGetType(ga) != genAlgTypeNeuraNet ||
    ga->_NNdata._nbIn != 1 ||
    ga->_NNdata._nbHid != 2 ||
    ga->_NNdata._nbOut != 3) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GASetTypeNeuraNet failed");
    PBErrCatch(GenAlgErr);
}
GenAlgFree(&ga);
printf("UnitTestGenAlgGetSet OK\n");

```

```

}

void UnitTestGenAlgInit() {
    srandom(5);
    int lengthAdnF = 2;
    int lengthAdnI = 2;
    GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
        lengthAdnF, lengthAdnI);
    VecFloat2D boundsF = VecFloatCreateStatic2D();
    VecLong2D boundsI = VecLongCreateStatic2D();
    VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
    VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
    GASetBoundsAdnFloat(ga, 0, &boundsF);
    GASetBoundsAdnFloat(ga, 1, &boundsF);
    GASetBoundsAdnInt(ga, 0, &boundsI);
    GASetBoundsAdnInt(ga, 1, &boundsI);
    GAInit(ga);
    GenAlgAdn* ent = (GenAlgAdn*)(GAAdns(ga)->_head->_data);
    if (ISEQUALF(VecGet(ent->_adnF, 0), -0.907064) == false ||
        ISEQUALF(VecGet(ent->_adnF, 1), -0.450509) == false ||
        VecGet(ent->_adnI, 0) != 2 ||
        VecGet(ent->_adnI, 1) != 10) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAInit failed");
        PBErrCatch(GenAlgErr);
    }
    GenAlgFree(&ga);
    printf("UnitTestGenAlgInit OK\n");
}

void UnitTestGenAlgPrint() {
    srandom(5);
    int lengthAdnF = 2;
    int lengthAdnI = 2;
    GenAlg* ga = GenAlgCreate(3, 2, lengthAdnF, lengthAdnI);
    VecFloat2D boundsF = VecFloatCreateStatic2D();
    VecLong2D boundsI = VecLongCreateStatic2D();
    VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
    VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
    GASetBoundsAdnFloat(ga, 0, &boundsF);
    GASetBoundsAdnFloat(ga, 1, &boundsF);
    GASetBoundsAdnInt(ga, 0, &boundsI);
    GASetBoundsAdnInt(ga, 1, &boundsI);
    GAInit(ga);
    GAPrintln(ga, stdout);
    GAEliteSummaryPrintln(ga, stdout);
    GenAlgFree(&ga);
    printf("UnitTestGenAlgInit OK\n");
}

void UnitTestGenAlgGetDiversity() {
    srandom(5);
    int lengthAdnF = 2;
    int lengthAdnI = 2;
    GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
        lengthAdnF, lengthAdnI);
    VecFloat2D boundsF = VecFloatCreateStatic2D();
    VecLong2D boundsI = VecLongCreateStatic2D();
    VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
    VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
    GASetBoundsAdnFloat(ga, 0, &boundsF);
    GASetBoundsAdnFloat(ga, 1, &boundsF);

```

```

    GASetBoundsAdnInt(ga, 0, &boundsI);
    GASetBoundsAdnInt(ga, 1, &boundsI);
    GASetNbElites(ga, 2);
    GASetNbEntities(ga, 3);
    GAInit(ga);
    if (ISEQUALF(GAGetDiversity(ga), 0.0) == false) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAGetDiversity failed");
        PBErrCatch(GenAlgErr);
    }
    VecCopy(GAAdn(ga, 1)->_adnF, GAAdn(ga, 0)->_adnF);
    VecCopy(GAAdn(ga, 1)->_adnI, GAAdn(ga, 0)->_adnI);
    if (ISEQUALF(GAGetDiversity(ga), 0.0) == false) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAGetDiversity failed");
        PBErrCatch(GenAlgErr);
    }
    GenAlgFree(&ga);
    printf("UnitTestGenAlgGetDiversity OK\n");
}

void UnitTestGenAlgStep() {
    srandom(2);
    int lengthAdnF = 2;
    int lengthAdnI = 2;
    GenAlg* ga = GenAlgCreate(3, 2, lengthAdnF, lengthAdnI);
    VecFloat2D boundsF = VecFloatCreateStatic2D();
    VecLong2D boundsI = VecLongCreateStatic2D();
    VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
    VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
    GASetBoundsAdnFloat(ga, 0, &boundsF);
    GASetBoundsAdnFloat(ga, 1, &boundsF);
    GASetBoundsAdnInt(ga, 0, &boundsI);
    GASetBoundsAdnInt(ga, 1, &boundsI);
    GAInit(ga);
    for (int i = 3; i--;)
        GASetAdnValue(ga, GAAdn(ga, i), 3.0 - (float)i);
    printf("Before Step:\n");
    GAPrintln(ga, stdout);
    GenAlgAdn* child = GAAdn(ga, 2);
    GAStep(ga);
    printf("After Step:\n");
    GAPrintln(ga, stdout);
    if (ga->_nextId != 4 || GAAdnGetId(child) != 3 ||
        GAAdnGetAge(child) != 1 ||
        ISEQUALF(GAAdnGetGeneF(child, 0), 0.285933) == false ||
        ISEQUALF(GAAdnGetGeneF(child, 1), 0.174965) == false ||
        ISEQUALF(GAAdnGetDeltaGeneF(child, 0), 0.0) == false ||
        ISEQUALF(GAAdnGetDeltaGeneF(child, 1), 0.0) == false ||
        GAAdnGetGeneI(child, 0) != 4 ||
        GAAdnGetGeneI(child, 1) != 10 ||
        GAAdn(ga, 2) != child ||
        GAAdnGetAge(GAAdn(ga, 0)) != 2 ||
        GAAdnGetAge(GAAdn(ga, 1)) != 2 ||
        GAAdnGetId(GAAdn(ga, 0)) != 0 ||
        GAAdnGetId(GAAdn(ga, 1)) != 1) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GAStep failed");
        PBErrCatch(GenAlgErr);
    }
    GenAlgFree(&ga);
    printf("UnitTestGenAlgStep OK\n");
}

```

```

}

void UnitTestGenAlgLoadSave() {
    srandom(5);
    int lengthAdnF = 2;
    int lengthAdnI = 2;
    GenAlg* ga = GenAlgCreate(3, 2, lengthAdnF, lengthAdnI);
    VecFloat2D boundsF = VecFloatCreateStatic2D();
    VecLong2D boundsI = VecLongCreateStatic2D();
    VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
    VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
    GASetBoundsAdnFloat(ga, 0, &boundsF);
    GASetBoundsAdnFloat(ga, 1, &boundsF);
    GASetBoundsAdnInt(ga, 0, &boundsI);
    GASetBoundsAdnInt(ga, 1, &boundsI);
    GAInit(ga);
    GASetStep(ga);
    GSet* rank = GSetCreate();
    for (int i = 3; i--;)
        GSetAddSort(rank, GAAdn(ga, i), 3.0 - (float)i);
    FILE* stream = fopen("./UnitTestGenAlgLoadSave.txt", "w");
    if (GASave(ga, stream, false) == false) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GASave failed");
        PBErrCatch(GenAlgErr);
    }
    fclose(stream);
    stream = fopen("./UnitTestGenAlgLoadSave.txt", "r");
    GenAlg* gaLoad = NULL;
    if (GALoad(&gaLoad, stream) == false) {
        GenAlgErr->_type = PBErrTypeUnitTestFailed;
        sprintf(GenAlgErr->_msg, "GALoad failed");
        PBErrCatch(GenAlgErr);
    }
    fclose(stream);
    if (ga->_nextId != gaLoad->_nextId ||
        ga->_curEpoch != gaLoad->_curEpoch ||
        ga->_nbElites != gaLoad->_nbElites ||
        ga->_type != genAlgTypeDefault ||
        ga->_lengthAdnF != gaLoad->_lengthAdnF ||
        ga->_lengthAdnI != gaLoad->_lengthAdnI ||
        VecIsEqual(ga->_boundsF, gaLoad->_boundsF) == false ||
        VecIsEqual(ga->_boundsF + 1, gaLoad->_boundsF + 1) == false ||
        VecIsEqual(ga->_boundsI, gaLoad->_boundsI) == false ||
        VecIsEqual(ga->_boundsI + 1, gaLoad->_boundsI + 1) == false ||
        GAAdnGetId(GAAdn(ga, 0)) != GAAdnGetId(GAAdn(gaLoad, 0)) ||
        GAAdnGetId(GAAdn(ga, 1)) != GAAdnGetId(GAAdn(gaLoad, 1)) ||
        GAAdnGetId(GAAdn(ga, 2)) != GAAdnGetId(GAAdn(gaLoad, 2)) ||
        GAAdnGetAge(GAAdn(ga, 0)) != GAAdnGetAge(GAAdn(gaLoad, 0)) ||
        GAAdnGetAge(GAAdn(ga, 1)) != GAAdnGetAge(GAAdn(gaLoad, 1)) ||
        GAAdnGetAge(GAAdn(ga, 2)) != GAAdnGetAge(GAAdn(gaLoad, 2)) ||
        VecIsEqual(GAAdn(ga, 0)->_adnF,
            GAAdn(gaLoad, 0)->_adnF) == false ||
        VecIsEqual(GAAdn(ga, 0)->_deltaAdnF,
            GAAdn(gaLoad, 0)->_deltaAdnF) == false ||
        VecIsEqual(GAAdn(ga, 0)->_adnI,
            GAAdn(gaLoad, 0)->_adnI) == false ||
        VecIsEqual(GAAdn(ga, 1)->_adnF,
            GAAdn(gaLoad, 1)->_adnF) == false ||
        VecIsEqual(GAAdn(ga, 1)->_deltaAdnF,
            GAAdn(gaLoad, 1)->_deltaAdnF) == false ||
        VecIsEqual(GAAdn(ga, 1)->_adnI,

```



```

        GAAdn(gaLoad, 1)->_adnI) == false ||
VecIsEqual(GAAdn(ga, 2)->_adnF,
        GAAdn(gaLoad, 2)->_adnF) == false ||
VecIsEqual(GAAdn(ga, 2)->_deltaAdnF,
        GAAdn(gaLoad, 2)->_deltaAdnF) == false ||
VecIsEqual(GAAdn(ga, 2)->_adnI,
        GAAdn(gaLoad, 2)->_adnI) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "UnitTestGenAlgLoadSave failed");
    PBErrCatch(GenAlgErr);
}
GSetFree(&rank);
GenAlgFree(&ga);
GenAlgFree(&gaLoad);
printf("UnitTestGenAlgLoadSave OK\n");
}

float ftarget(float x) {
    return -0.5 * fastpow(x, 3) + 0.314 * fastpow(x, 2) - 0.7777 * x + 0.1;
}

float evaluate(const VecFloat* adnF, const VecLong* adnI) {
    float delta = 0.02;
    int nb = (int)round(4.0 / delta);
    float res = 0.0;
    float x = -2.0;
    for (int i = 0; i < nb; ++i, x += delta) {
        float y = 0.0;
        for (int j = 4; j--;)
            y += VecGet(adnF, j) * fastpow(x, VecGet(adnI, j));
        res += fabs(ftarget(x) - y);
    }
    return res / (float)nb;
}

void UnitTestGenAlgTest() {
    srandom(0);
    int lengthAdnF = 4;
    int lengthAdnI = lengthAdnF;
    GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
        lengthAdnF, lengthAdnI);
    VecFloat2D boundsF = VecFloatCreateStatic2D();
    VecLong2D boundsI = VecLongCreateStatic2D();
    VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
    VecSet(&boundsI, 0, 0); VecSet(&boundsI, 1, 4);
    for (int i = lengthAdnF; i--;) {
        GSetBoundsAdnFloat(ga, i, &boundsF);
        GSetBoundsAdnInt(ga, i, &boundsI);
    }
    GAINit(ga);
    GSetTextOMeterFlag(ga, true);
    //GSetDiversityThreshold(ga, 0.0001);
    GSetDiversityThreshold(ga, 0.01);
    GSetNbMinAdn(ga, 32);
    GSetNbMaxAdn(ga, 512);
    float best = 1.0;
    do {
        for (int iEnt = GAGetNbAdns(ga); iEnt--;)
            if (GAAdnIsNew(GAAdn(ga, iEnt)))
                GSetAdnValue(ga, GAAdn(ga, iEnt),
                    -1.0 * evaluate(GAAdnAdnF(GAAdn(ga, iEnt)),
                        GAAdnAdnI(GAAdn(ga, iEnt))));
    }
}

```

```

    GAStep(ga);
    // Slow down the process to have time to read the TextOMeter
    unsigned int microseconds = 10000;
    usleep(microseconds);
    //sleep(1);
    // Display info if there is improvment
    float ev = evaluate(GABestAdnF(ga), GABestAdnI(ga));
    if (best - ev > PBMath_EPSILON) {
        best = ev;
        printf("%lu %f ", GAGetCurEpoch(ga), best);
        VecFloatPrint(GABestAdnF(ga), stdout, 6);
        printf(" ");
        VecPrint(GABestAdnI(ga), stdout);
        printf("\n");
    }
} while (GAGetCurEpoch(ga) < 2000 && best > PBMath_EPSILON);
printf("target: -0.5*x^3 + 0.314*x^2 - 0.7777*x + 0.1\n");
printf("approx: \n");
GAAdnPrintln(GABestAdn(ga), stdout);
printf("error: %f\n", evaluate(GABestAdnF(ga), GABestAdnI(ga)));
GenAlgFree(&ga);
printf("UnitTestGenAlgTest OK\n");
}

void UnitTestGenAlgPerf() {
    int nbRun = 10;
    unsigned long int nbMaxEpoch = 2000;
    float maxEv = 0.0;
    float bestEv = 0.0;
    float sumEv = 0.0;
    float avgEv = 0.0;
    for (int iRun = 0; iRun < nbRun; ++iRun) {
        srandom(time(NULL));
        int lengthAdnF = 4;
        int lengthAdnI = lengthAdnF;
        GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
            lengthAdnF, lengthAdnI);
        VecFloat2D boundsF = VecFloatCreateStatic2D();
        VecLong2D boundsI = VecLongCreateStatic2D();
        VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
        VecSet(&boundsI, 0, 0); VecSet(&boundsI, 1, 4);
        for (int i = lengthAdnF; i--;) {
            GASetBoundsAdnFloat(ga, i, &boundsF);
            GASetBoundsAdnInt(ga, i, &boundsI);
        }
        GAInit(ga);
        //GASetDiversityThreshold(ga, 0.001);

        GASetDiversityThreshold(ga, 0.01);
        GASetNbMinAdn(ga, 32);
        GASetNbMaxAdn(ga, 512);

        float ev = 0.0;
        do {
            for (int iEnt = GAGetNbAdns(ga); iEnt--;)
                if (GAAdnIsNew(GAAdn(ga, iEnt)))
                    GASetAdnValue(ga, GAAdn(ga, iEnt),
                        -1.0 * evaluate(GAAdnAdnF(GAAdn(ga, iEnt)),
                            GAAdnAdnI(GAAdn(ga, iEnt))));
            GAStep(ga);
            ev = evaluate(GABestAdnF(ga), GABestAdnI(ga));
        } while (GAGetCurEpoch(ga) < nbMaxEpoch && ev > PBMath_EPSILON);
    }
}

```

```

        sumEv += ev;
        if (iRun == 0 || bestEv > ev)
            bestEv = ev;
        if (iRun == 0 || maxEv < ev)
            maxEv = ev;
        avgEv = sumEv / (float)(iRun + 1);
        printf("best: %f, worst: %f, avg: %f, ktevent: %lu\n",
            bestEv, maxEv, avgEv, ga->_nbKTEvent);
        GenAlgFree(&ga);
    }
    avgEv = sumEv / (float)nbRun;
    printf("in %d runs, %lu epochs, best: %f, worst: %f, avg: %f\n",
        nbRun, nbMaxEpoch, bestEv, maxEv, avgEv);
    printf("UnitTestGenAlgPerf OK\n");
}

void UnitTestGenAlg() {
    UnitTestGenAlgCreateFree();
    UnitTestGenAlgGetSet();
    UnitTestGenAlgInit();
    UnitTestGenAlgPrint();
    UnitTestGenAlgGetDiversity();
    UnitTestGenAlgStep();
    UnitTestGenAlgLoadSave();
    UnitTestGenAlgTest();
    UnitTestGenAlgPerf();
    printf("UnitTestGenAlg OK\n");
}

void UnitTestAll() {
    UnitTestGenAlgAdn();
    UnitTestGenAlg();
    printf("UnitTestAll OK\n");
}

int main() {
    UnitTestAll();
    // Return success code
    return 0;
}

```

## 6 Unit tests output

```

UnitTestGenAlgAdnCreateFree OK
UnitTestGenAlgAdnGetSet OK
UnitTestGenAlgAdnInit OK
UnitTestGenAlgAdn OK
UnitTestGenAlgCreateFree OK
UnitTestGenAlgGetSet OK
UnitTestGenAlgInit OK
epoch:0
3 entities, 2 elites
#0 value:0.000000 elite id:0 age:1
  adnF:<0.788004,-0.003504>
  deltaAdnF:<0.000000,0.000000>
  adnI:<3,1>
#1 value:0.000000 elite id:1 age:1
  adnF:<-0.840711,-0.704622>

```

```

    deltaAdnF:<0.000000,0.000000>
    adnI:<5,4>
#2 value:0.000000 id:2 age:1
    adnF:<-0.907064,-0.450509>
    deltaAdnF:<0.000000,0.000000>
    adnI:<2,10>
(age,val,div) (1,0.000,0.000) (1,0.000,0.455)
UnitTestGenAlgInit OK
UnitTestGenAlgGetDiversity OK
Before Step:
epoch:0
3 entities, 2 elites
#0 value:3.000000 elite id:0 age:1
    adnF:<0.285933,0.174965>
    deltaAdnF:<0.000000,0.000000>
    adnI:<4,10>
#1 value:2.000000 elite id:1 age:1
    adnF:<-0.156076,-0.303386>
    deltaAdnF:<0.000000,0.000000>
    adnI:<2,7>
#2 value:1.000000 id:2 age:1
    adnF:<0.619353,0.401953>
    deltaAdnF:<0.000000,0.000000>
    adnI:<2,2>
After Step:
epoch:1
3 entities, 2 elites
#0 value:3.000000 elite id:0 age:2
    adnF:<0.285933,0.174965>
    deltaAdnF:<0.000000,0.000000>
    adnI:<4,10>
#1 value:2.000000 elite id:1 age:2
    adnF:<-0.156076,-0.303386>
    deltaAdnF:<0.000000,0.000000>
    adnI:<2,7>
#2 value:1.000000 id:3 age:1
    adnF:<0.285933,0.174965>
    deltaAdnF:<0.000000,0.000000>
    adnI:<4,10>
UnitTestGenAlgStep OK
UnitTestGenAlgLoadSave OK
2 0.202594 <-0.556069,-0.411679,-0.262673,0.499542> <3,1,1,2>
4 0.101712 <-0.524388,-0.198029,0.317662,-0.496964> <1,1,2,3>
6 0.094902 <-0.710292,0.018952,0.372442,-0.556692> <1,2,2,3>
9 0.083031 <-0.556069,-0.646450,0.372442,-0.016666> <3,1,2,0>
10 0.079733 <-0.836658,0.018952,0.372442,-0.468830> <1,2,2,3>
11 0.068616 <-0.505736,-0.785797,0.372442,0.085225> <3,1,2,0>
15 0.063889 <-0.836658,0.089179,0.274934,-0.468830> <1,2,2,3>
19 0.063198 <-0.855053,0.089179,0.274934,-0.468830> <1,2,2,3>
23 0.061650 <-0.848098,0.089179,0.274934,-0.468830> <1,2,2,3>
24 0.061242 <-0.848098,0.084148,0.274934,-0.468830> <1,2,2,3>
27 0.060846 <-0.851505,0.084148,0.274934,-0.468830> <1,2,2,3>
31 0.060517 <-0.718301,0.008625,0.351759,-0.514215> <1,0,2,3>
33 0.055760 <-0.718301,0.077882,0.351759,-0.544232> <1,0,2,3>
34 0.055732 <-0.718301,0.008625,0.351759,-0.520389> <1,0,2,3>
38 0.048148 <-0.718301,0.033906,0.351759,-0.520389> <1,0,2,3>
39 0.045611 <-0.718301,0.091931,0.289099,-0.520389> <1,0,2,3>
45 0.035618 <-0.718301,0.140258,0.289099,-0.520389> <1,0,2,3>
55 0.023134 <-0.743370,0.117274,0.317665,-0.515054> <1,0,2,3>
56 0.022043 <-0.743370,0.115488,0.298783,-0.519234> <1,0,2,3>
59 0.019943 <-0.743370,0.115488,0.298783,-0.515054> <1,0,2,3>
64 0.018260 <-0.761297,0.114413,0.298783,-0.513328> <1,0,2,3>

```

```

66 0.017453 <-0.755398,0.114413,0.298783,-0.513328> <1,0,2,3>
75 0.016354 <-0.760443,0.114413,0.298783,-0.508511> <1,0,2,3>
83 0.015222 <-0.776573,0.114413,0.298783,-0.504213> <1,0,2,3>
84 0.010572 <-0.776573,0.096385,0.309279,-0.504213> <1,0,2,3>
94 0.010069 <-0.789251,0.096385,0.309279,-0.495285> <1,0,2,3>
102 0.010007 <-0.786850,0.096385,0.309279,-0.495285> <1,0,2,3>
110 0.007566 <-0.790270,0.102254,0.309279,-0.495285> <1,0,2,3>
112 0.006426 <-0.786850,0.102254,0.309279,-0.495285> <1,0,2,3>
123 0.006338 <-0.783914,0.102254,0.309279,-0.495285> <1,0,2,3>
185 0.006325 <-0.783914,0.102322,0.309279,-0.495285> <1,0,2,3>
187 0.006051 <-0.782752,0.101120,0.315162,-0.495285> <1,0,2,3>
192 0.005528 <-0.788900,0.102254,0.315162,-0.495285> <1,0,2,3>
196 0.005136 <-0.788427,0.101120,0.315162,-0.495285> <1,0,2,3>
206 0.005043 <-0.786692,0.101120,0.315162,-0.495285> <1,0,2,3>
212 0.004795 <-0.786692,0.098350,0.315162,-0.495285> <1,0,2,3>
291 0.003860 <-0.786692,0.098350,0.315162,-0.496681> <1,0,2,3>
465 0.003597 <-0.782872,0.098350,0.315162,-0.496681> <1,0,2,3>
597 0.003515 <-0.782872,0.098350,0.315162,-0.499100> <1,0,2,3>
635 0.003427 <-0.782872,0.099359,0.315162,-0.499100> <1,0,2,3>
642 0.002565 <-0.782872,0.099359,0.315162,-0.498289> <1,0,2,3>
649 0.002467 <-0.782872,0.099359,0.315162,-0.498104> <1,0,2,3>
674 0.002089 <-0.781099,0.099359,0.315162,-0.498104> <1,0,2,3>
691 0.002008 <-0.774996,0.099359,0.315162,-0.501728> <1,0,2,3>
864 0.001865 <-0.774996,0.099359,0.314146,-0.501728> <1,0,2,3>
874 0.001744 <-0.774996,0.099359,0.314146,-0.501640> <1,0,2,3>
1308 0.001240 <-0.774996,0.099359,0.314521,-0.501094> <1,0,2,3>
1646 0.001221 <-0.775075,0.099359,0.314521,-0.501094> <1,0,2,3>
1649 0.001170 <-0.775432,0.099359,0.314521,-0.501094> <1,0,2,3>
2268 0.001158 <-0.775524,0.099359,0.314521,-0.501084> <1,0,2,3>
3890 0.001124 <-0.775524,0.099442,0.314307,-0.501084> <1,0,2,3>
4136 0.001114 <-0.775524,0.099791,0.314307,-0.501084> <1,0,2,3>
5250 0.001098 <-0.775524,0.099791,0.314307,-0.500583> <1,0,2,3>
5858 0.001033 <-0.775524,0.099791,0.314118,-0.500583> <1,0,2,3>
7729 0.000915 <-0.775524,0.099791,0.314118,-0.500727> <1,0,2,3>
target: -0.5*x^3 + 0.314*x^2 - 0.7777*x + 0.1
approx:
id:5500898 age:7729
  adnF:<-0.775524,0.099791,0.314118,-0.500727>
  deltaAdnF:<-0.218328,-0.013918,-0.138814,-0.007539>
  adnI:<1,0,2,3>
error: 0.000915
UnitTestGenAlgTest OK
best: 0.001139, worst: 0.001139, avg: inf, ktevent: 437941
best: 0.000366, worst: 0.001139, avg: 0.001505, ktevent: 442756
best: 0.000366, worst: 0.001246, avg: 0.001375, ktevent: 439416
best: 0.000366, worst: 0.001246, avg: 0.001293, ktevent: 438333
best: 0.000366, worst: 0.001246, avg: 0.001230, ktevent: 441636
best: 0.000350, worst: 0.001246, avg: 0.001054, ktevent: 439725
best: 0.000350, worst: 0.001246, avg: 0.000962, ktevent: 438441
best: 0.000350, worst: 0.001246, avg: 0.000924, ktevent: 438800
best: 0.000350, worst: 0.001246, avg: 0.000915, ktevent: 442410
best: 0.000350, worst: 0.001246, avg: 0.000917, ktevent: 442881
best: 0.000350, worst: 0.001246, avg: 0.000943, ktevent: 441871
best: 0.000350, worst: 0.001246, avg: 0.000938, ktevent: 440597
best: 0.000350, worst: 0.001246, avg: 0.000923, ktevent: 433294
best: 0.000350, worst: 0.001246, avg: 0.000882, ktevent: 441164
best: 0.000350, worst: 0.002092, avg: 0.000968, ktevent: 439625
best: 0.000350, worst: 0.002092, avg: 0.000992, ktevent: 438065
best: 0.000350, worst: 0.002092, avg: 0.000964, ktevent: 438098
best: 0.000350, worst: 0.002092, avg: 0.000955, ktevent: 441497
best: 0.000350, worst: 0.002092, avg: 0.000998, ktevent: 440108
best: 0.000350, worst: 0.002092, avg: 0.000972, ktevent: 442243

```

best: 0.000143, worst: 0.002092, avg: 0.000931, ktevent: 439819  
 best: 0.000143, worst: 0.002092, avg: 0.000905, ktevent: 438779  
 best: 0.000143, worst: 0.002092, avg: 0.000898, ktevent: 440382  
 best: 0.000143, worst: 0.002092, avg: 0.000887, ktevent: 439790  
 best: 0.000143, worst: 0.002092, avg: 0.000879, ktevent: 443273  
 best: 0.000143, worst: 0.002092, avg: 0.000857, ktevent: 439070  
 best: 0.000143, worst: 0.002092, avg: 0.000837, ktevent: 442204  
 best: 0.000143, worst: 0.002092, avg: 0.000825, ktevent: 438290  
 best: 0.000143, worst: 0.002092, avg: 0.000818, ktevent: 443657  
 best: 0.000143, worst: 0.002092, avg: 0.000817, ktevent: 442966  
 best: 0.000143, worst: 0.002092, avg: 0.000827, ktevent: 444890  
 best: 0.000143, worst: 0.002092, avg: 0.000834, ktevent: 444375  
 best: 0.000143, worst: 0.002092, avg: 0.000829, ktevent: 439432  
 best: 0.000143, worst: 0.002092, avg: 0.000823, ktevent: 440238  
 best: 0.000143, worst: 0.002092, avg: 0.000832, ktevent: 442338  
 best: 0.000143, worst: 0.002092, avg: 0.000825, ktevent: 440603  
 best: 0.000143, worst: 0.002092, avg: 0.000826, ktevent: 442803  
 best: 0.000143, worst: 0.002092, avg: 0.000823, ktevent: 436893  
 best: 0.000143, worst: 0.002092, avg: 0.000848, ktevent: 440124  
 best: 0.000143, worst: 0.002092, avg: 0.000835, ktevent: 434704  
 best: 0.000143, worst: 0.002092, avg: 0.000824, ktevent: 436028  
 best: 0.000143, worst: 0.002092, avg: 0.000818, ktevent: 440100  
 best: 0.000143, worst: 0.002092, avg: 0.000821, ktevent: 440420  
 best: 0.000143, worst: 0.002092, avg: 0.000833, ktevent: 440627  
 best: 0.000143, worst: 0.002092, avg: 0.000853, ktevent: 442387  
 best: 0.000114, worst: 0.002092, avg: 0.000836, ktevent: 440545  
 best: 0.000114, worst: 0.002571, avg: 0.000874, ktevent: 440148  
 best: 0.000114, worst: 0.002571, avg: 0.000885, ktevent: 440131  
 best: 0.000114, worst: 0.002571, avg: 0.000890, ktevent: 441830  
 best: 0.000114, worst: 0.002571, avg: 0.000894, ktevent: 438493  
 best: 0.000114, worst: 0.002571, avg: 0.000899, ktevent: 443494  
 best: 0.000114, worst: 0.002571, avg: 0.000890, ktevent: 442616  
 best: 0.000114, worst: 0.002571, avg: 0.000891, ktevent: 440626  
 best: 0.000114, worst: 0.002571, avg: 0.000890, ktevent: 433764  
 best: 0.000114, worst: 0.002571, avg: 0.000883, ktevent: 437589  
 best: 0.000114, worst: 0.002571, avg: 0.000890, ktevent: 439944  
 best: 0.000114, worst: 0.002571, avg: 0.000890, ktevent: 441088  
 best: 0.000114, worst: 0.002571, avg: 0.000879, ktevent: 443107  
 best: 0.000114, worst: 0.002571, avg: 0.000884, ktevent: 442255  
 best: 0.000114, worst: 0.002571, avg: 0.000873, ktevent: 442044  
 best: 0.000114, worst: 0.002571, avg: 0.000871, ktevent: 436747  
 best: 0.000114, worst: 0.002571, avg: 0.000861, ktevent: 440131  
 best: 0.000114, worst: 0.002571, avg: 0.000851, ktevent: 439880  
 best: 0.000114, worst: 0.002571, avg: 0.000853, ktevent: 442178  
 best: 0.000114, worst: 0.002571, avg: 0.000858, ktevent: 438272  
 best: 0.000114, worst: 0.002571, avg: 0.000852, ktevent: 442290  
 best: 0.000114, worst: 0.002571, avg: 0.000860, ktevent: 444937  
 best: 0.000114, worst: 0.002571, avg: 0.000856, ktevent: 438130  
 best: 0.000114, worst: 0.002571, avg: 0.000851, ktevent: 440617  
 best: 0.000114, worst: 0.002571, avg: 0.000841, ktevent: 440705  
 best: 0.000114, worst: 0.002571, avg: 0.000849, ktevent: 439393  
 best: 0.000114, worst: 0.002571, avg: 0.000850, ktevent: 436922  
 best: 0.000114, worst: 0.002571, avg: 0.000843, ktevent: 439223  
 best: 0.000114, worst: 0.002571, avg: 0.000840, ktevent: 440808  
 best: 0.000114, worst: 0.002571, avg: 0.000836, ktevent: 446770  
 best: 0.000114, worst: 0.002571, avg: 0.000841, ktevent: 438190  
 best: 0.000114, worst: 0.002571, avg: 0.000856, ktevent: 439630  
 best: 0.000114, worst: 0.002571, avg: 0.000861, ktevent: 439639  
 best: 0.000114, worst: 0.002571, avg: 0.000859, ktevent: 439802  
 best: 0.000114, worst: 0.002571, avg: 0.000850, ktevent: 441432  
 best: 0.000114, worst: 0.002571, avg: 0.000855, ktevent: 437091  
 best: 0.000114, worst: 0.002571, avg: 0.000872, ktevent: 438972

best: 0.000114, worst: 0.002571, avg: 0.000873, ktevent: 441247  
best: 0.000114, worst: 0.002571, avg: 0.000875, ktevent: 438896  
best: 0.000114, worst: 0.002571, avg: 0.000869, ktevent: 442940  
best: 0.000114, worst: 0.002571, avg: 0.000874, ktevent: 437802  
best: 0.000114, worst: 0.002571, avg: 0.000874, ktevent: 440017  
best: 0.000114, worst: 0.002571, avg: 0.000873, ktevent: 441841  
best: 0.000114, worst: 0.002571, avg: 0.000879, ktevent: 439068  
best: 0.000114, worst: 0.002571, avg: 0.000874, ktevent: 441243  
best: 0.000114, worst: 0.002571, avg: 0.000886, ktevent: 440718  
best: 0.000114, worst: 0.002571, avg: 0.000883, ktevent: 439244  
best: 0.000114, worst: 0.002571, avg: 0.000878, ktevent: 440757  
best: 0.000114, worst: 0.002571, avg: 0.000874, ktevent: 440819  
best: 0.000114, worst: 0.002571, avg: 0.000878, ktevent: 435278  
best: 0.000114, worst: 0.002571, avg: 0.000878, ktevent: 438802  
best: 0.000114, worst: 0.002571, avg: 0.000873, ktevent: 441768  
best: 0.000114, worst: 0.002571, avg: 0.000875, ktevent: 439600  
best: 0.000114, worst: 0.002571, avg: 0.000873, ktevent: 442285  
best: 0.000114, worst: 0.002571, avg: 0.000879, ktevent: 440260  
best: 0.000114, worst: 0.002571, avg: 0.000880, ktevent: 439717  
best: 0.000114, worst: 0.002571, avg: 0.000875, ktevent: 441894  
best: 0.000114, worst: 0.002571, avg: 0.000876, ktevent: 440824  
best: 0.000114, worst: 0.002571, avg: 0.000879, ktevent: 438331  
best: 0.000114, worst: 0.002571, avg: 0.000884, ktevent: 438750  
best: 0.000114, worst: 0.002571, avg: 0.000879, ktevent: 440870  
best: 0.000114, worst: 0.002571, avg: 0.000879, ktevent: 437107  
best: 0.000114, worst: 0.002571, avg: 0.000884, ktevent: 440013  
best: 0.000114, worst: 0.002571, avg: 0.000878, ktevent: 442129  
best: 0.000114, worst: 0.002571, avg: 0.000875, ktevent: 438058  
best: 0.000114, worst: 0.002571, avg: 0.000876, ktevent: 439610  
best: 0.000114, worst: 0.002571, avg: 0.000870, ktevent: 440731  
best: 0.000114, worst: 0.002571, avg: 0.000867, ktevent: 441731  
best: 0.000114, worst: 0.002571, avg: 0.000874, ktevent: 442405  
best: 0.000114, worst: 0.002571, avg: 0.000878, ktevent: 438038  
best: 0.000114, worst: 0.002571, avg: 0.000875, ktevent: 440130  
best: 0.000114, worst: 0.002571, avg: 0.000874, ktevent: 438462  
best: 0.000114, worst: 0.002571, avg: 0.000873, ktevent: 443145  
best: 0.000114, worst: 0.002571, avg: 0.000873, ktevent: 441815  
best: 0.000114, worst: 0.002571, avg: 0.000871, ktevent: 440006  
best: 0.000114, worst: 0.002571, avg: 0.000872, ktevent: 438702  
best: 0.000114, worst: 0.002571, avg: 0.000876, ktevent: 438381  
best: 0.000114, worst: 0.002571, avg: 0.000874, ktevent: 441722  
best: 0.000114, worst: 0.002571, avg: 0.000872, ktevent: 440307  
best: 0.000114, worst: 0.002571, avg: 0.000869, ktevent: 439732  
best: 0.000114, worst: 0.002571, avg: 0.000864, ktevent: 440887  
best: 0.000114, worst: 0.002571, avg: 0.000862, ktevent: 441223  
best: 0.000114, worst: 0.002571, avg: 0.000860, ktevent: 437473  
best: 0.000114, worst: 0.002571, avg: 0.000860, ktevent: 445019  
best: 0.000114, worst: 0.002571, avg: 0.000862, ktevent: 438678  
best: 0.000114, worst: 0.002571, avg: 0.000862, ktevent: 442808  
best: 0.000114, worst: 0.002571, avg: 0.000865, ktevent: 441622  
best: 0.000114, worst: 0.002571, avg: 0.000865, ktevent: 442788  
best: 0.000114, worst: 0.002571, avg: 0.000866, ktevent: 438082  
best: 0.000114, worst: 0.002571, avg: 0.000862, ktevent: 434795  
best: 0.000114, worst: 0.002571, avg: 0.000859, ktevent: 438128  
best: 0.000114, worst: 0.002571, avg: 0.000855, ktevent: 441032  
best: 0.000114, worst: 0.002571, avg: 0.000850, ktevent: 441117  
best: 0.000114, worst: 0.002571, avg: 0.000846, ktevent: 441550  
best: 0.000114, worst: 0.002571, avg: 0.000846, ktevent: 441279  
best: 0.000114, worst: 0.002571, avg: 0.000844, ktevent: 441238  
best: 0.000114, worst: 0.002571, avg: 0.000842, ktevent: 437568  
best: 0.000114, worst: 0.002571, avg: 0.000845, ktevent: 438443  
best: 0.000114, worst: 0.002571, avg: 0.000846, ktevent: 440958

best: 0.000114, worst: 0.002571, avg: 0.000850, ktevent: 440643  
 best: 0.000114, worst: 0.002571, avg: 0.000853, ktevent: 437443  
 best: 0.000114, worst: 0.002571, avg: 0.000853, ktevent: 441132  
 best: 0.000114, worst: 0.002571, avg: 0.000852, ktevent: 439313  
 best: 0.000114, worst: 0.002571, avg: 0.000856, ktevent: 439296  
 best: 0.000114, worst: 0.002571, avg: 0.000855, ktevent: 440542  
 best: 0.000114, worst: 0.002571, avg: 0.000859, ktevent: 439993  
 best: 0.000114, worst: 0.002571, avg: 0.000863, ktevent: 438598  
 best: 0.000114, worst: 0.002571, avg: 0.000863, ktevent: 441566  
 best: 0.000114, worst: 0.002571, avg: 0.000862, ktevent: 442267  
 best: 0.000114, worst: 0.002571, avg: 0.000862, ktevent: 438901  
 best: 0.000114, worst: 0.002571, avg: 0.000860, ktevent: 439563  
 best: 0.000114, worst: 0.002571, avg: 0.000858, ktevent: 442840  
 best: 0.000114, worst: 0.002571, avg: 0.000856, ktevent: 438849  
 best: 0.000114, worst: 0.002571, avg: 0.000855, ktevent: 440540  
 best: 0.000114, worst: 0.002571, avg: 0.000856, ktevent: 440312  
 best: 0.000114, worst: 0.002571, avg: 0.000861, ktevent: 443918  
 best: 0.000114, worst: 0.002571, avg: 0.000861, ktevent: 442645  
 best: 0.000114, worst: 0.002571, avg: 0.000863, ktevent: 438825  
 best: 0.000104, worst: 0.002571, avg: 0.000858, ktevent: 438785  
 best: 0.000104, worst: 0.002571, avg: 0.000862, ktevent: 437864  
 best: 0.000104, worst: 0.002571, avg: 0.000859, ktevent: 442747  
 best: 0.000104, worst: 0.002571, avg: 0.000862, ktevent: 440690  
 best: 0.000104, worst: 0.002571, avg: 0.000862, ktevent: 441680  
 best: 0.000104, worst: 0.002571, avg: 0.000865, ktevent: 445894  
 best: 0.000104, worst: 0.002571, avg: 0.000866, ktevent: 439207  
 best: 0.000104, worst: 0.002571, avg: 0.000861, ktevent: 437867  
 best: 0.000104, worst: 0.002571, avg: 0.000858, ktevent: 439077  
 best: 0.000104, worst: 0.002571, avg: 0.000859, ktevent: 439736  
 best: 0.000104, worst: 0.002571, avg: 0.000857, ktevent: 440049  
 best: 0.000104, worst: 0.002571, avg: 0.000860, ktevent: 438037  
 best: 0.000104, worst: 0.002571, avg: 0.000860, ktevent: 440096  
 best: 0.000104, worst: 0.002571, avg: 0.000859, ktevent: 441152  
 best: 0.000104, worst: 0.002571, avg: 0.000856, ktevent: 438233  
 best: 0.000104, worst: 0.002571, avg: 0.000858, ktevent: 438115  
 best: 0.000104, worst: 0.002571, avg: 0.000856, ktevent: 439512  
 best: 0.000104, worst: 0.002571, avg: 0.000855, ktevent: 438697  
 best: 0.000104, worst: 0.002571, avg: 0.000856, ktevent: 439240  
 best: 0.000104, worst: 0.002571, avg: 0.000856, ktevent: 439667  
 best: 0.000104, worst: 0.002571, avg: 0.000855, ktevent: 439572  
 best: 0.000104, worst: 0.002571, avg: 0.000857, ktevent: 436472  
 best: 0.000104, worst: 0.002571, avg: 0.000857, ktevent: 434467  
 best: 0.000104, worst: 0.002571, avg: 0.000854, ktevent: 438339  
 best: 0.000104, worst: 0.002571, avg: 0.000857, ktevent: 442343  
 best: 0.000104, worst: 0.002571, avg: 0.000860, ktevent: 441798  
 best: 0.000104, worst: 0.002571, avg: 0.000860, ktevent: 440378  
 best: 0.000104, worst: 0.002571, avg: 0.000860, ktevent: 443495  
 best: 0.000104, worst: 0.002571, avg: 0.000858, ktevent: 439713  
 best: 0.000104, worst: 0.002571, avg: 0.000857, ktevent: 442354  
 best: 0.000104, worst: 0.002571, avg: 0.000858, ktevent: 435048  
 best: 0.000104, worst: 0.002571, avg: 0.000859, ktevent: 441376  
 best: 0.000104, worst: 0.002571, avg: 0.000861, ktevent: 438297  
 best: 0.000104, worst: 0.002767, avg: 0.000870, ktevent: 442442  
 best: 0.000104, worst: 0.002767, avg: 0.000870, ktevent: 440143  
 best: 0.000104, worst: 0.002767, avg: 0.000869, ktevent: 442941  
 best: 0.000104, worst: 0.002767, avg: 0.000869, ktevent: 439934  
 best: 0.000104, worst: 0.002767, avg: 0.000869, ktevent: 443037  
 best: 0.000104, worst: 0.002767, avg: 0.000870, ktevent: 442280  
 best: 0.000104, worst: 0.002767, avg: 0.000872, ktevent: 439061  
 best: 0.000104, worst: 0.002767, avg: 0.000870, ktevent: 436451  
 best: 0.000104, worst: 0.002767, avg: 0.000871, ktevent: 441953  
 best: 0.000104, worst: 0.002767, avg: 0.000876, ktevent: 440997



best: 0.000104, worst: 0.002767, avg: 0.000879, ktevent: 440189  
 best: 0.000104, worst: 0.002767, avg: 0.000886, ktevent: 434734  
 best: 0.000104, worst: 0.002767, avg: 0.000885, ktevent: 438309  
 best: 0.000104, worst: 0.002767, avg: 0.000888, ktevent: 440754  
 best: 0.000104, worst: 0.002767, avg: 0.000886, ktevent: 436575  
 best: 0.000104, worst: 0.002767, avg: 0.000884, ktevent: 440944  
 best: 0.000104, worst: 0.002767, avg: 0.000886, ktevent: 440517  
 best: 0.000104, worst: 0.002767, avg: 0.000883, ktevent: 440730  
 best: 0.000104, worst: 0.002767, avg: 0.000891, ktevent: 440536  
 best: 0.000104, worst: 0.002767, avg: 0.000889, ktevent: 443916  
 best: 0.000104, worst: 0.002767, avg: 0.000891, ktevent: 440213  
 best: 0.000104, worst: 0.002767, avg: 0.000890, ktevent: 438852  
 best: 0.000066, worst: 0.002767, avg: 0.000886, ktevent: 441375  
 best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 440058  
 best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 441469  
 best: 0.000066, worst: 0.002767, avg: 0.000896, ktevent: 440785  
 best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 439270  
 best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 439939  
 best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 442702  
 best: 0.000066, worst: 0.002767, avg: 0.000891, ktevent: 443098  
 best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 439393  
 best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 438302  
 best: 0.000066, worst: 0.002767, avg: 0.000898, ktevent: 435480  
 best: 0.000066, worst: 0.002767, avg: 0.000898, ktevent: 440722  
 best: 0.000066, worst: 0.002767, avg: 0.000897, ktevent: 442299  
 best: 0.000066, worst: 0.002767, avg: 0.000896, ktevent: 436612  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 443337  
 best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 436136  
 best: 0.000066, worst: 0.002767, avg: 0.000891, ktevent: 435815  
 best: 0.000066, worst: 0.002767, avg: 0.000891, ktevent: 438202  
 best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 443718  
 best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 439502  
 best: 0.000066, worst: 0.002767, avg: 0.000894, ktevent: 442425  
 best: 0.000066, worst: 0.002767, avg: 0.000896, ktevent: 443631  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 441322  
 best: 0.000066, worst: 0.002767, avg: 0.000894, ktevent: 441233  
 best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 434957  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 439400  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 442883  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 441039  
 best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 440974  
 best: 0.000066, worst: 0.002767, avg: 0.000894, ktevent: 437787  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 442104  
 best: 0.000066, worst: 0.002767, avg: 0.000896, ktevent: 441035  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 439621  
 best: 0.000066, worst: 0.002767, avg: 0.000894, ktevent: 441259  
 best: 0.000066, worst: 0.002767, avg: 0.000894, ktevent: 437694  
 best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 442341  
 best: 0.000066, worst: 0.002767, avg: 0.000894, ktevent: 441226  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 442146  
 best: 0.000066, worst: 0.002767, avg: 0.000898, ktevent: 441109  
 best: 0.000066, worst: 0.002767, avg: 0.000896, ktevent: 442585  
 best: 0.000066, worst: 0.002767, avg: 0.000896, ktevent: 436087  
 best: 0.000066, worst: 0.002767, avg: 0.000901, ktevent: 439747  
 best: 0.000066, worst: 0.002767, avg: 0.000900, ktevent: 440868  
 best: 0.000066, worst: 0.002767, avg: 0.000901, ktevent: 438437  
 best: 0.000066, worst: 0.002767, avg: 0.000897, ktevent: 443411  
 best: 0.000066, worst: 0.002767, avg: 0.000897, ktevent: 440158  
 best: 0.000066, worst: 0.002767, avg: 0.000898, ktevent: 442291  
 best: 0.000066, worst: 0.002767, avg: 0.000898, ktevent: 439234  
 best: 0.000066, worst: 0.002767, avg: 0.000897, ktevent: 437413  
 best: 0.000066, worst: 0.002767, avg: 0.000897, ktevent: 440306

best: 0.000066, worst: 0.002767, avg: 0.000897, ktevent: 441763  
 best: 0.000066, worst: 0.002767, avg: 0.000896, ktevent: 440061  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 437636  
 best: 0.000066, worst: 0.002767, avg: 0.000894, ktevent: 438267  
 best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 435976  
 best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 444031  
 best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 441541  
 best: 0.000066, worst: 0.002767, avg: 0.000896, ktevent: 440811  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 440903  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 439478  
 best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 440991  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 438674  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 439341  
 best: 0.000066, worst: 0.002767, avg: 0.000902, ktevent: 440363  
 best: 0.000066, worst: 0.002767, avg: 0.000903, ktevent: 438651  
 best: 0.000066, worst: 0.002767, avg: 0.000903, ktevent: 442060  
 best: 0.000066, worst: 0.002767, avg: 0.000905, ktevent: 444983  
 best: 0.000066, worst: 0.002767, avg: 0.000906, ktevent: 437424  
 best: 0.000066, worst: 0.002767, avg: 0.000908, ktevent: 441891  
 best: 0.000066, worst: 0.002767, avg: 0.000906, ktevent: 440963  
 best: 0.000066, worst: 0.002767, avg: 0.000904, ktevent: 439642  
 best: 0.000066, worst: 0.002767, avg: 0.000905, ktevent: 441546  
 best: 0.000066, worst: 0.002767, avg: 0.000904, ktevent: 438529  
 best: 0.000066, worst: 0.002767, avg: 0.000902, ktevent: 442362  
 best: 0.000066, worst: 0.002767, avg: 0.000903, ktevent: 438796  
 best: 0.000066, worst: 0.002767, avg: 0.000904, ktevent: 443045  
 best: 0.000066, worst: 0.002767, avg: 0.000905, ktevent: 443177  
 best: 0.000066, worst: 0.002767, avg: 0.000907, ktevent: 440894  
 best: 0.000066, worst: 0.002767, avg: 0.000905, ktevent: 440279  
 best: 0.000066, worst: 0.002767, avg: 0.000907, ktevent: 437627  
 best: 0.000066, worst: 0.002767, avg: 0.000907, ktevent: 439900  
 best: 0.000066, worst: 0.002767, avg: 0.000908, ktevent: 437457  
 best: 0.000066, worst: 0.002767, avg: 0.000908, ktevent: 439946  
 best: 0.000066, worst: 0.002767, avg: 0.000908, ktevent: 439420  
 best: 0.000066, worst: 0.002767, avg: 0.000907, ktevent: 438153  
 best: 0.000066, worst: 0.002767, avg: 0.000905, ktevent: 440636  
 best: 0.000066, worst: 0.002767, avg: 0.000903, ktevent: 438997  
 best: 0.000066, worst: 0.002767, avg: 0.000904, ktevent: 442593  
 best: 0.000066, worst: 0.002767, avg: 0.000904, ktevent: 440789  
 best: 0.000066, worst: 0.002767, avg: 0.000903, ktevent: 438938  
 best: 0.000066, worst: 0.002767, avg: 0.000901, ktevent: 440431  
 best: 0.000066, worst: 0.002767, avg: 0.000900, ktevent: 442224  
 best: 0.000066, worst: 0.002767, avg: 0.000897, ktevent: 437459  
 best: 0.000066, worst: 0.002767, avg: 0.000897, ktevent: 438238  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 439548  
 best: 0.000066, worst: 0.002767, avg: 0.000894, ktevent: 441250  
 best: 0.000066, worst: 0.002767, avg: 0.000894, ktevent: 441690  
 best: 0.000066, worst: 0.002767, avg: 0.000894, ktevent: 441419  
 best: 0.000066, worst: 0.002767, avg: 0.000896, ktevent: 441496  
 best: 0.000066, worst: 0.002767, avg: 0.000896, ktevent: 441899  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 441527  
 best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 439172  
 best: 0.000066, worst: 0.002767, avg: 0.000894, ktevent: 439955  
 best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 441165  
 best: 0.000066, worst: 0.002767, avg: 0.000891, ktevent: 435457  
 best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 441489  
 best: 0.000066, worst: 0.002767, avg: 0.000888, ktevent: 442321  
 best: 0.000066, worst: 0.002767, avg: 0.000888, ktevent: 444006  
 best: 0.000066, worst: 0.002767, avg: 0.000889, ktevent: 442896  
 best: 0.000066, worst: 0.002767, avg: 0.000888, ktevent: 436561  
 best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 443577  
 best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 438986

best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 440632  
 best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 441359  
 best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 441460  
 best: 0.000066, worst: 0.002767, avg: 0.000889, ktevent: 442645  
 best: 0.000066, worst: 0.002767, avg: 0.000889, ktevent: 440428  
 best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 442219  
 best: 0.000066, worst: 0.002767, avg: 0.000891, ktevent: 438424  
 best: 0.000066, worst: 0.002767, avg: 0.000889, ktevent: 443027  
 best: 0.000066, worst: 0.002767, avg: 0.000888, ktevent: 446966  
 best: 0.000066, worst: 0.002767, avg: 0.000887, ktevent: 440814  
 best: 0.000066, worst: 0.002767, avg: 0.000889, ktevent: 439718  
 best: 0.000066, worst: 0.002767, avg: 0.000889, ktevent: 437097  
 best: 0.000066, worst: 0.002767, avg: 0.000891, ktevent: 440791  
 best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 442705  
 best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 436945  
 best: 0.000066, worst: 0.002767, avg: 0.000891, ktevent: 440791  
 best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 443943  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 439456  
 best: 0.000066, worst: 0.002767, avg: 0.000894, ktevent: 436807  
 best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 437492  
 best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 439537  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 441915  
 best: 0.000066, worst: 0.002767, avg: 0.000895, ktevent: 439951  
 best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 439878  
 best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 440984  
 best: 0.000066, worst: 0.002767, avg: 0.000894, ktevent: 442931  
 best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 441253  
 best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 438126  
 best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 435710  
 best: 0.000066, worst: 0.002767, avg: 0.000889, ktevent: 438338  
 best: 0.000066, worst: 0.002767, avg: 0.000888, ktevent: 440615  
 best: 0.000066, worst: 0.002767, avg: 0.000886, ktevent: 442595  
 best: 0.000066, worst: 0.002767, avg: 0.000885, ktevent: 438369  
 best: 0.000066, worst: 0.002767, avg: 0.000885, ktevent: 438225  
 best: 0.000066, worst: 0.002767, avg: 0.000883, ktevent: 442787  
 best: 0.000066, worst: 0.002767, avg: 0.000882, ktevent: 442426  
 best: 0.000066, worst: 0.002767, avg: 0.000881, ktevent: 435266  
 best: 0.000066, worst: 0.002767, avg: 0.000880, ktevent: 441715  
 best: 0.000066, worst: 0.002767, avg: 0.000880, ktevent: 439269  
 best: 0.000066, worst: 0.002767, avg: 0.000878, ktevent: 442219  
 best: 0.000066, worst: 0.002767, avg: 0.000877, ktevent: 437737  
 best: 0.000066, worst: 0.002767, avg: 0.000879, ktevent: 440434  
 best: 0.000066, worst: 0.002767, avg: 0.000879, ktevent: 441217  
 best: 0.000066, worst: 0.002767, avg: 0.000879, ktevent: 442493  
 best: 0.000066, worst: 0.002767, avg: 0.000880, ktevent: 441442  
 best: 0.000066, worst: 0.002767, avg: 0.000879, ktevent: 439569  
 best: 0.000066, worst: 0.002767, avg: 0.000878, ktevent: 443354  
 best: 0.000066, worst: 0.002767, avg: 0.000877, ktevent: 437638  
 best: 0.000066, worst: 0.002767, avg: 0.000876, ktevent: 437649  
 best: 0.000066, worst: 0.002767, avg: 0.000876, ktevent: 441944  
 best: 0.000066, worst: 0.002767, avg: 0.000876, ktevent: 440945  
 best: 0.000066, worst: 0.002767, avg: 0.000875, ktevent: 442756  
 best: 0.000066, worst: 0.002767, avg: 0.000876, ktevent: 442350  
 best: 0.000066, worst: 0.002767, avg: 0.000875, ktevent: 444126  
 best: 0.000066, worst: 0.002767, avg: 0.000875, ktevent: 438838  
 best: 0.000066, worst: 0.002767, avg: 0.000874, ktevent: 438946  
 best: 0.000066, worst: 0.002767, avg: 0.000874, ktevent: 434124  
 best: 0.000066, worst: 0.002767, avg: 0.000872, ktevent: 444374  
 best: 0.000066, worst: 0.002767, avg: 0.000871, ktevent: 437961  
 best: 0.000066, worst: 0.002767, avg: 0.000871, ktevent: 440187  
 best: 0.000066, worst: 0.002767, avg: 0.000870, ktevent: 437864  
 best: 0.000066, worst: 0.002767, avg: 0.000869, ktevent: 444819

best: 0.000066, worst: 0.002767, avg: 0.000870, ktevent: 441694  
 best: 0.000066, worst: 0.002767, avg: 0.000868, ktevent: 439039  
 best: 0.000066, worst: 0.002767, avg: 0.000869, ktevent: 442512  
 best: 0.000066, worst: 0.002767, avg: 0.000868, ktevent: 439212  
 best: 0.000066, worst: 0.002767, avg: 0.000870, ktevent: 438959  
 best: 0.000066, worst: 0.002767, avg: 0.000869, ktevent: 438083  
 best: 0.000066, worst: 0.002767, avg: 0.000869, ktevent: 443309  
 best: 0.000066, worst: 0.002767, avg: 0.000870, ktevent: 440415  
 best: 0.000066, worst: 0.002767, avg: 0.000871, ktevent: 438810  
 best: 0.000066, worst: 0.002767, avg: 0.000873, ktevent: 442421  
 best: 0.000066, worst: 0.002767, avg: 0.000873, ktevent: 438454  
 best: 0.000066, worst: 0.002767, avg: 0.000872, ktevent: 438681  
 best: 0.000066, worst: 0.002767, avg: 0.000871, ktevent: 442068  
 best: 0.000066, worst: 0.002767, avg: 0.000870, ktevent: 440669  
 best: 0.000066, worst: 0.002767, avg: 0.000871, ktevent: 439160  
 best: 0.000066, worst: 0.002767, avg: 0.000871, ktevent: 440809  
 best: 0.000066, worst: 0.002767, avg: 0.000871, ktevent: 443346  
 best: 0.000066, worst: 0.002767, avg: 0.000870, ktevent: 442558  
 best: 0.000066, worst: 0.002767, avg: 0.000871, ktevent: 443426  
 best: 0.000066, worst: 0.002767, avg: 0.000869, ktevent: 440023  
 best: 0.000066, worst: 0.002767, avg: 0.000870, ktevent: 439324  
 best: 0.000066, worst: 0.002767, avg: 0.000869, ktevent: 443873  
 best: 0.000066, worst: 0.002767, avg: 0.000867, ktevent: 442456  
 best: 0.000066, worst: 0.002767, avg: 0.000871, ktevent: 442515  
 best: 0.000066, worst: 0.002767, avg: 0.000870, ktevent: 439278  
 best: 0.000066, worst: 0.002767, avg: 0.000871, ktevent: 440929  
 best: 0.000066, worst: 0.002767, avg: 0.000873, ktevent: 443445  
 best: 0.000066, worst: 0.002767, avg: 0.000874, ktevent: 442574  
 best: 0.000066, worst: 0.002767, avg: 0.000873, ktevent: 440142  
 best: 0.000066, worst: 0.002767, avg: 0.000873, ktevent: 442380  
 best: 0.000066, worst: 0.002767, avg: 0.000873, ktevent: 440164  
 best: 0.000066, worst: 0.002767, avg: 0.000876, ktevent: 440812  
 best: 0.000066, worst: 0.002767, avg: 0.000876, ktevent: 438196  
 best: 0.000066, worst: 0.002767, avg: 0.000876, ktevent: 443102  
 best: 0.000066, worst: 0.002767, avg: 0.000877, ktevent: 438489  
 best: 0.000066, worst: 0.002767, avg: 0.000876, ktevent: 437382  
 best: 0.000066, worst: 0.002767, avg: 0.000876, ktevent: 440537  
 best: 0.000066, worst: 0.002767, avg: 0.000876, ktevent: 436418  
 best: 0.000066, worst: 0.002767, avg: 0.000879, ktevent: 440858  
 best: 0.000066, worst: 0.002767, avg: 0.000881, ktevent: 440091  
 best: 0.000066, worst: 0.002767, avg: 0.000881, ktevent: 439606  
 best: 0.000066, worst: 0.002767, avg: 0.000883, ktevent: 437453  
 best: 0.000066, worst: 0.002767, avg: 0.000884, ktevent: 443519  
 best: 0.000066, worst: 0.002767, avg: 0.000884, ktevent: 438366  
 best: 0.000066, worst: 0.002767, avg: 0.000885, ktevent: 439416  
 best: 0.000066, worst: 0.002767, avg: 0.000886, ktevent: 442649  
 best: 0.000066, worst: 0.002767, avg: 0.000885, ktevent: 438363  
 best: 0.000066, worst: 0.002767, avg: 0.000885, ktevent: 440492  
 best: 0.000066, worst: 0.002767, avg: 0.000884, ktevent: 438126  
 best: 0.000066, worst: 0.002767, avg: 0.000884, ktevent: 439624  
 best: 0.000066, worst: 0.002767, avg: 0.000883, ktevent: 440944  
 best: 0.000066, worst: 0.002767, avg: 0.000883, ktevent: 440032  
 best: 0.000066, worst: 0.002767, avg: 0.000883, ktevent: 437638  
 best: 0.000066, worst: 0.002767, avg: 0.000882, ktevent: 445241  
 best: 0.000066, worst: 0.002767, avg: 0.000881, ktevent: 442306  
 best: 0.000066, worst: 0.002767, avg: 0.000880, ktevent: 438969  
 best: 0.000066, worst: 0.002767, avg: 0.000880, ktevent: 437032  
 best: 0.000066, worst: 0.002767, avg: 0.000879, ktevent: 442859  
 best: 0.000066, worst: 0.002767, avg: 0.000880, ktevent: 438724  
 best: 0.000066, worst: 0.002767, avg: 0.000881, ktevent: 443544  
 best: 0.000066, worst: 0.002767, avg: 0.000882, ktevent: 440595  
 best: 0.000066, worst: 0.002767, avg: 0.000882, ktevent: 441589

```

best: 0.000066, worst: 0.002767, avg: 0.000882, ktevent: 441675
best: 0.000066, worst: 0.002767, avg: 0.000881, ktevent: 441641
best: 0.000066, worst: 0.002767, avg: 0.000880, ktevent: 439743
best: 0.000066, worst: 0.002767, avg: 0.000881, ktevent: 441498
best: 0.000066, worst: 0.002767, avg: 0.000882, ktevent: 437789
best: 0.000066, worst: 0.002767, avg: 0.000883, ktevent: 440061
best: 0.000066, worst: 0.002767, avg: 0.000884, ktevent: 440339
best: 0.000066, worst: 0.002767, avg: 0.000884, ktevent: 440344
best: 0.000066, worst: 0.002767, avg: 0.000883, ktevent: 444698
best: 0.000066, worst: 0.002767, avg: 0.000885, ktevent: 436936
best: 0.000066, worst: 0.002767, avg: 0.000884, ktevent: 442867
best: 0.000066, worst: 0.002767, avg: 0.000885, ktevent: 440735
best: 0.000066, worst: 0.002767, avg: 0.000885, ktevent: 440146
best: 0.000066, worst: 0.002767, avg: 0.000886, ktevent: 439761
best: 0.000066, worst: 0.002767, avg: 0.000885, ktevent: 437368
best: 0.000066, worst: 0.002767, avg: 0.000888, ktevent: 441216
best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 441937
best: 0.000066, worst: 0.002767, avg: 0.000889, ktevent: 437543
best: 0.000066, worst: 0.002767, avg: 0.000887, ktevent: 441593
best: 0.000066, worst: 0.002767, avg: 0.000888, ktevent: 443043
best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 439958
best: 0.000066, worst: 0.002767, avg: 0.000891, ktevent: 441513
best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 444577
best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 445178
best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 439463
best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 442339
best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 438774
best: 0.000066, worst: 0.002767, avg: 0.000894, ktevent: 440173
best: 0.000066, worst: 0.002767, avg: 0.000894, ktevent: 440805
best: 0.000066, worst: 0.002767, avg: 0.000894, ktevent: 439691
best: 0.000066, worst: 0.002767, avg: 0.000893, ktevent: 439106
best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 442196
best: 0.000066, worst: 0.002767, avg: 0.000891, ktevent: 441274
best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 440073
best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 442264
best: 0.000066, worst: 0.002767, avg: 0.000891, ktevent: 441124
best: 0.000066, worst: 0.002767, avg: 0.000892, ktevent: 441874
best: 0.000066, worst: 0.002767, avg: 0.000891, ktevent: 439659
best: 0.000066, worst: 0.002767, avg: 0.000891, ktevent: 439143
best: 0.000066, worst: 0.002767, avg: 0.000891, ktevent: 441004
best: 0.000066, worst: 0.002767, avg: 0.000891, ktevent: 442090
best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 438783
best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 442893
best: 0.000066, worst: 0.002767, avg: 0.000889, ktevent: 439195
best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 438797
best: 0.000066, worst: 0.002767, avg: 0.000890, ktevent: 437336
in 500 runs, 2000 epochs, best: 0.000066, worst: 0.002767, avg: 0.000888
UnitTestGenAlgPerf OK
UnitTestGenAlg OK
UnitTestAll OK

```

UnitTestGenAlgLoadSave.txt:

```

{
  "_type": "0",
  "_nbAdns": "3",
  "_nbElites": "2",
  "_lengthAdnF": "2",
  "_lengthAdnI": "2",
  "_curEpoch": "1",
  "_nextId": "4",

```

```

"_boundFloat": [
  {
    "_dim": "2",
    "_val": ["-1.000000", "1.000000"]
  },
  {
    "_dim": "2",
    "_val": ["-1.000000", "1.000000"]
  }
],
"_boundInt": [
  {
    "_dim": "2",
    "_val": ["1", "10"]
  },
  {
    "_dim": "2",
    "_val": ["1", "10"]
  }
],
"_adns": [
  {
    "_id": "3",
    "_age": "1",
    "_elo": "0.000000",
    "_val": "0.000000",
    "_adnF": {
      "_dim": "2",
      "_val": ["0.788004", "-0.003504"]
    },
    "_deltaAdnF": {
      "_dim": "2",
      "_val": ["0.000000", "0.000000"]
    },
    "_adnI": {
      "_dim": "2",
      "_val": ["3", "1"]
    }
  },
  {
    "_id": "1",
    "_age": "2",
    "_elo": "0.000000",
    "_val": "0.000000",
    "_adnF": {
      "_dim": "2",
      "_val": ["-0.840711", "-0.704622"]
    },
    "_deltaAdnF": {
      "_dim": "2",
      "_val": ["0.000000", "0.000000"]
    },
    "_adnI": {
      "_dim": "2",
      "_val": ["5", "4"]
    }
  },
  {
    "_id": "0",
    "_age": "2",
    "_elo": "0.000000",
    "_val": "0.000000",

```

```

    "_adnF":{
      "_dim":"2",
      "_val":["0.788004","-0.003504"]
    },
    "_deltaAdnF":{
      "_dim":"2",
      "_val":["0.000000","0.000000"]
    },
    "_adnI":{
      "_dim":"2",
      "_val":["3","1"]
    }
  }
],
"_bestAdn":{
  "_id":"0",
  "_age":"1",
  "_elo":"0.000000",
  "_val":"0.000000",
  "_adnF":{
    "_dim":"2",
    "_val":["0.788004","-0.003504"]
  },
  "_deltaAdnF":{
    "_dim":"2",
    "_val":["0.000000","0.000000"]
  },
  "_adnI":{
    "_dim":"2",
    "_val":["3","1"]
  }
}
}

```

eval() of best genes over epoch:

