GenAlg

P. Baillehache

March 17, 2018

Contents

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. GenAlg uses the library `ELORank` to manage the selection of entities during the reproduction phase. 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`, `GSet` and `ELORank` 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 hte 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: $P = \frac{rank}{nbEntity}$ where rank is the rank of the discarded entity in the pool of entities, and nbEntity is the number of entities in the pool. 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), we also apply mutation to all the elite entities except the best one when the inbreeding level of the elite pool fall below a threshold set to 0.1. The mutation of elites occurs exactly as the one for discarded entities, but the age of elite entity itself is used instead of the one of selected entities (which doesn't exist in this case). The inbreeding level is calculated as follow $\frac{1}{nbElite} \sum_{i=1}^{nbElite} \frac{||\overrightarrow{adn}(elite_i) - \overrightarrow{adn}(elite_0)||}{||\overrightarrow{bound}_{max} - \overrightarrow{bound}_{min}||}$ where nbElite is the number of elite entities, $\overrightarrow{adn}(elite_i)$ is the genes vector of the i-th elite entity, and $\overrightarrow{bound}_{max}$ and $\overrightarrow{bound}_{min}$ 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 posibilities 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 "elorank.h"

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

#define GENALG_RUNPEREPOCH 100
#define GENALG_NBENTITIES 100
#define GENALG_NBELITES 20
#define GENALG_INBREEDINGTHRESHOLD 0.1

// ------------- GenAlgEntity

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

typedef struct GenAlg GenAlg;

typedef struct GenAlgEntity {
  // ID
  int _id;
  // Age
  int _age;
  // Adn for floating point value
  VecFloat* _adnF;
  // Delta Adn during mutation
  VecFloat* _deltaAdnF;
  // Adn for integer point value
  VecShort* _adnI;
} GenAlgEntity;

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

// Create a new GenAlgEntity with ID 'id', 'lengthAdnF' and 'lengthAdnI'
// 'lengthAdnF' and 'lengthAdnI' must be greater than or equal to 0
GenAlgEntity* GenAlgEntityCreate(int id, int lengthAdnF,
  int lengthAdnI);

// Free memory used by the GenAlgEntity 'that'
void GenAlgEntityFree(GenAlgEntity** that);

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

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

// Return the adn for integer values of the GenAlgEntity 'that'
#if BUILDMODE != 0
inline
#endif
VecShort* GAEntAdnI(GenAlgEntity* that);
```

```
// Initialise randomly the genes of the GenAlgEntity 'that' of the
// GenAlg 'ga'
void GAEntInit(GenAlgEntity* that, GenAlg* ga);

// Get the 'iGene'-th gene of the adn for floating point values of the
// GenAlgEntity 'that'
#if BUILDMODE != 0
inline
#endif
float GAEntGetGeneF(GenAlgEntity* that, int iGene);

// Get the delta of the 'iGene'-th gene of the adn for floating point
// values of the GenAlgEntity 'that'
#if BUILDMODE != 0
inline
#endif
float GAEntGetDeltaGeneF(GenAlgEntity* that, int iGene);

// Get the 'iGene'-th gene of the adn for int values of the
// GenAlgEntity 'that'
#if BUILDMODE != 0
inline
#endif
int GAEntGetGeneI(GenAlgEntity* that, int iGene);

// Set the 'iGene'-th gene of the adn for floating point values of the
// GenAlgEntity 'that' to 'gene'
#if BUILDMODE != 0
inline
#endif
void GAEntSetGeneF(GenAlgEntity* that, int iGene, float gene);

// Set the delta of the 'iGene'-th gene of the adn for floating point
// values of the GenAlgEntity 'that' to 'delta'
#if BUILDMODE != 0
inline
#endif
void GAEntSetDeltaGeneF(GenAlgEntity* that, int iGene, float delta);

// Set the 'iGene'-th gene of the adn for int values of the
// GenAlgEntity 'that'to 'gene'
#if BUILDMODE != 0
inline
#endif
void GAEntSetGeneI(GenAlgEntity* that, int iGene, short gene);

// Get the id of the GenAlgEntity 'that'
#if BUILDMODE != 0
inline
#endif
int GAEntGetId(GenAlgEntity* that);

// Get the age of the GenAlgEntity 'that'
#if BUILDMODE != 0
inline
#endif
int GAEntGetAge(GenAlgEntity* that);

// Print the information about the GenAlgEntity 'that' on the
// stream 'stream'
void GAEntPrintln(GenAlgEntity* that, FILE* stream);
```

```
// ------------- GenAlg

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

typedef struct GenAlg {
  // ELORank of GenAlgEntity
  ELORank* _elo;
  // Nb runs per epoch
  int _runsPerEpoch;
  // Current run
  int _curRun;
  // Current epoch
  int _curEpoch;
  // Nb entities in population
  int _nbEntities;
  // Nb elite entities in population
  int _nbElites;
  // Id of the next new GenAlgEntity
  int _nextId;
  // Length of adn for floating point value
  int _lengthAdnF;
  // Length of adn for integer value
  int _lengthAdnI;
  // Bounds (min, max) for floating point values adn
  VecFloat2D* _boundsF;
  // Bounds (min, max) for integer values adn
  VecShort2D* _boundsI;
} 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(int nbEntities, int nbElites, int lengthAdnF,
  int lengthAdnI);

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

// Return the ELORank of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
ELORank* GAEloRank(GenAlg* that);

// Return the nb of runs per epoch of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
int GAGetRunsPerEpoch(GenAlg* that);

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

// Return the nb of elites of the GenAlg 'that'
```

```
#if BUILDMODE != 0
inline
#endif
int GAGetNbElites(GenAlg* that);

// Return the current run of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
int GAGetCurRun(GenAlg* that);

// Return the current epoch of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
int GAGetCurEpoch(GenAlg* that);

// Set the nb of runs per epoch of the GenAlg 'that' to 'runs'
// 'runs' must be greater than 0
#if BUILDMODE != 0
inline
#endif
void GASetRunsPerEpoch(GenAlg* that, int runs);

// 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* that, 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* that, int nb);

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

// Get the length of adn for integer value
#if BUILDMODE != 0
inline
#endif
int GAGetLengthAdnInt(GenAlg* that);

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

// Get the bounds for the 'iGene'-th gene of adn for integer values
#if BUILDMODE != 0
inline
#endif
VecShort2D* GABoundsAdnInt(GenAlg* that, int iGene);

// Get the GenAlgEntity of the GenAlg 'that' currently at rank 'iRank'
#if BUILDMODE != 0
inline
```

```
#endif
GenAlgEntity* GAEntity(GenAlg* that, 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* that);

// Step a run for the GenAlg 'that' with ranking of GenAlgEntity given
// in the GSet of GenAlgEntity 'rank' (from best to worst, ie _sortVal
// from greater to lower)
void GAStepRun(GenAlg* that, GSet* rank);

// Step an epoch for the GenAlg 'that' with the current ranking of
// GenAlgEntity
void GAStepEpoch(GenAlg* that);

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

// Get the level of inbreeding of curent entities of the GenAlg 'that'
// The return value is in [0.0, 1.0]
// 0.0 means all the elite entities have exactly the same adns
float GAGetInbreeding(GenAlg* that);

// 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* stream);

// Save the GenAlg 'that' to the stream 'stream'
// Return true in case of success, else false
bool GASave(GenAlg* that, FILE* stream);

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

// ================ Inliner ====================

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


#endif
```

3 Code

3.1 genalg.c

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

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

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

```
// ------------- GenAlgEntity

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

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

// Mute the genes of the entity at rank 'iChild'
// The probability of mutation for one gene is equal to
// 'iChild'/'that'->_nbEntities and the amplitude of the mutation
// is equal to (max-min).(gauss(0.0, 1.0)+deltaAdn).ln('parents[0]'.age)
void GAMute(GenAlg* that, int* parents, int iChild);

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

// Create a new GenAlgEntity with ID 'id', 'lengthAdnF' and 'lengthAdnI'
// 'lengthAdnF' and 'lengthAdnI' must be greater than or equal to 0
GenAlgEntity* GenAlgEntityCreate(int id, int lengthAdnF,
  int lengthAdnI) {
#if BUILDMODE == 0
  if (lengthAdnF < 0) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'lengthAdnF' is invalid (%d>=0)",
      lengthAdnF);
    PBErrCatch(GenAlgErr);
  }
  if (lengthAdnI < 0) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'lengthAdnI' is invalid (%d>=0)",
      lengthAdnI);
    PBErrCatch(GenAlgErr);
  }
#endif
  // Allocate memory
  GenAlgEntity* that = PBErrMalloc(GenAlgErr, sizeof(GenAlgEntity));
  // Set the properties
  that->_age = 1;
  that->_id = id;
  if (lengthAdnF > 0) {
    that->_adnF = VecFloatCreate(lengthAdnF);
    that->_deltaAdnF = VecFloatCreate(lengthAdnF);
  } else {
    that->_adnF = NULL;
    that->_deltaAdnF = NULL;
  }
  if (lengthAdnI > 0)
    that->_adnI = VecShortCreate(lengthAdnI);
  else
    that->_adnI = NULL;
  // Return the new GenAlgEntity
  return that;
}

// Free memory used by the GenAlgEntity 'that'
void GenAlgEntityFree(GenAlgEntity** 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));
  free(*that);
  // Set the pointer to null
  *that = NULL;
}

// Initialise randomly the genes of the GenAlgEntity 'that' of the
// GenAlg 'ga'
void GAEntInit(GenAlgEntity* that, GenAlg* 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 (int 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);
  }
  // For each integer value gene
  for (int iGene = GAGetLengthAdnInt(ga); iGene--;) {
    short min = VecGet(GABoundsAdnInt(ga, iGene), 0);
    short max = VecGet(GABoundsAdnInt(ga, iGene), 1);
    short val = (short)round((float)min + (float)(max - min) * rnd());
    VecSet(that->_adnI, iGene, val);
  }
}

// Print the information about the GenAlgEntity 'that' on the
// stream 'stream'
void GAEntPrintln(GenAlgEntity* that, FILE* 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:%d age:%d", GAEntGetId(that), GAEntGetAge(that));
  fprintf(stream, "\n");
  fprintf(stream, "  adnF:");
  VecPrint(GAEntAdnF(that), stream);
  fprintf(stream, "\n");
  fprintf(stream, "  deltaAdnF:");
  VecPrint(GAEntDeltaAdnF(that), stream);
  fprintf(stream, "\n");
```

```
  fprintf(stream, "  adnI:");
  VecPrint(GAEntAdnI(that), stream);
  fprintf(stream, "\n");
}

// ------------- GenAlg

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

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

// 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(int nbEntities, int nbElites, int lengthAdnF,
  int lengthAdnI) {
  // Allocate memory
  GenAlg* that = PBErrMalloc(GenAlgErr, sizeof(GenAlg));
  // Set the properties
  that->_elo = ELORankCreate();
  that->_runsPerEpoch = GENALG_RUNPEREPOCH;
  that->_curRun = 0;
  that->_curEpoch = 0;
  that->_lengthAdnF = lengthAdnF;
  that->_lengthAdnI = lengthAdnI;
  if (lengthAdnF > 0) {
    that->_boundsF =
      PBErrMalloc(GenAlgErr, sizeof(VecFloat2D) * lengthAdnF);
    for (int iGene = lengthAdnF; iGene--;)
      that->_boundsF[iGene] = VecFloatCreateStatic2D();
  } else
    that->_boundsF = NULL;
  if (lengthAdnI > 0) {
    that->_boundsI =
      PBErrMalloc(GenAlgErr, sizeof(VecShort2D) * lengthAdnI);
    for (int iGene = lengthAdnI; iGene--;)
      that->_boundsI[iGene] = VecShortCreateStatic2D();
  } else
    that->_boundsI = NULL;
  that->_nbEntities = 0;
  that->_nbElites = 0;
  that->_nextId = 0;
  GASetNbEntities(that, nbEntities);
  GASetNbElites(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
  for (int iEnt = (*that)->_nbEntities; iEnt--;) {
    GenAlgEntity* gaEnt = GAEntity(*that, iEnt);
    GenAlgEntityFree(&gaEnt);
  }
  ELORankFree(&((*that)->_elo));
  if ((*that)->_boundsF != NULL)
    free((*that)->_boundsF);
```

```
  if ((*that)->_boundsI != NULL)
    free((*that)->_boundsI);
  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* that, int nb) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (nb <= 1) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'nb' is invalid (%d>1)", nb);
    PBErrCatch(GenAlgErr);
  }
#endif
  while (that->_nbEntities > nb) {
    ELOEntity* ent = ELORankGetRanked(GAEloRank(that),
      that->_nbEntities - 1);
    GenAlgEntity* gaEnt = (GenAlgEntity*)(ent->_data);
    ELORankRemove(GAEloRank(that), ent->_data);
    GenAlgEntityFree(&gaEnt);
    that->_nbEntities = ELORankGetNb(GAEloRank(that));
  }
  while (that->_nbEntities < nb) {
    GenAlgEntity* ent = GenAlgEntityCreate(that->_nextId++,
      GAGetLengthAdnFloat(that), GAGetLengthAdnInt(that));
    ELORankAdd(GAEloRank(that), ent);
    that->_nbEntities = ELORankGetNb(GAEloRank(that));
  }
  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* that, int nb) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (nb <= 1) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'nb' is invalid (%d>1)", nb);
    PBErrCatch(GenAlgErr);
  }
#endif
  if (GAGetNbEntities(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 GAInit(GenAlg* that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  // For each entity
  GSetIterForward iter =
    GSetIterForwardCreateStatic(&(GAEloRank(that)->_set));
  do {
    // Get the entity
    GenAlgEntity* ent = ((ELOEntity*)GSetIterGet(&iter))->_data;
    // Initialise randomly the genes of the entity
    GAEntInit(ent, that);
  } while (GSetIterStep(&iter));
}

// Step a run for the GenAlg 'that' with ranking of GenAlgEntity given
// in the GSet of GenAlgEntity 'rank'  (from best to worst, ie _sortVal
// from greater to lower)
void GAStepRun(GenAlg* that, GSet* rank) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (rank == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'rank' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  // Update the ELORank
  ELORankUpdate(GAEloRank(that), rank);
  // Increment the number of runs
  ++(that->_curRun);
  // Increment the age of the entities of this run
  GSetIterForward iter = GSetIterForwardCreateStatic(rank);
  do {
    GenAlgEntity* ent = GSetIterGet(&iter);
    ++(ent->_age);
  } while (GSetIterStep(&iter));
  // If we have reached the end of the current epoch
  if (that->_curRun >= that->_runsPerEpoch)
    // Step the epoch
    GAStepEpoch(that);
}

// Step an epoch for the GenAlg 'that' with the current ranking of
// GenAlgEntity
void GAStepEpoch(GenAlg* that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
```

```
    PBErrCatch(GenAlgErr);
  }
#endif
  // Selection, Reproduction, Mutation
  // Declare a GSet of GenAlgEntity to memorize the childs
  GSet* childs = GSetCreate();
  // Declare a variable to memorize the parents
  int parents[2];
  // Get the inbreeding level
  float inbreeding = GAGetInbreeding(that);
  // If the inbreeding level is too high
  // Break the inbreeding by applying mutation to elites except the
  // best one
  if (inbreeding < GENALG_INBREEDINGTHRESHOLD) {
    // For each entity which is not an elite
    for (int iEnt = 1; iEnt < GAGetNbElites(that); ++iEnt) {
      // Reproduce with itself and mute the genes of the entity
      parents[1] = parents[0] = iEnt;
      GAReproduction(that, parents, iEnt);
      GAMute(that, parents, iEnt);
      // Add this entity to the set of childs too
      GSetAppend(childs, GAEntity(that, iEnt));
    }
  }
  // For each entity which is not an elite
  for (int iEnt = GAGetNbElites(that); iEnt < GAGetNbEntities(that);
    ++iEnt) {
    // Select two parents for this entity
    GASelectParents(that, parents);
    // Set the genes of the entity as a 50/50 mix of parents' genes
    GAReproduction(that, parents, iEnt);
    // Mute the genes of the entity
    GAMute(that, parents, iEnt);
    // Add the child to the set of childs
    GSetAppend(childs, GAEntity(that, iEnt));
  }
  // Remove and re-add the childs from/to the ELOrank to reset their
  // position and ELO
  GSetIterForward iter = GSetIterForwardCreateStatic(childs);
  do {
    ELORankRemove(GAEloRank(that), GSetIterGet(&iter));
    ELORankAdd(GAEloRank(that), GSetIterGet(&iter));
  } while (GSetIterStep(&iter));
  // Increment the number of epochs
  ++(that->_curEpoch);
  // Reset the current run
  that->_curRun = 0;
  // Free memory
  GSetFree(&childs);
}

// Select the rank of two parents for the SRM algorithm
// Return the ranks in 'parents', with parents[0] <= parents[1]
void GASelectParents(GenAlg* that, int* parents) {
#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);
  }
#endif
  // Declare a variable to memorize the parents' rank
  int p[2];
  for (int i = 2; i--;)
    // p[i] below may be equal to the rank of the highest non elite
    // entity, 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));
  // 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 entity at rank 'iChild' as a 50/50 mix of the
// genes of entities at ranks 'parents[0]' and 'parents[1]'
void GAReproduction(GenAlg* that, int* parents, 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 >= that->_nbEntities) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<%d)",
      iChild, that->_nbEntities);
    PBErrCatch(GenAlgErr);
  }
#endif
  // Get the parents and child
  GenAlgEntity* parentA = GAEntity(that, parents[0]);
  GenAlgEntity* parentB = GAEntity(that, parents[1]);
  GenAlgEntity* child = GAEntity(that, iChild);
  // For each gene of the adn for floating point value
  for (int iGene = GAGetLengthAdnFloat(that); iGene--;) {
    // Get the gene from one parent or the other with equal probabililty
    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 (int iGene = GAGetLengthAdnInt(that); iGene--;) {
```

```
    // Get the gene from one parent or the other with equal probabililty
    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)++;
}

// Mute the genes of the entity at rank 'iChild'
// The probability of mutation for one gene is equal to
// 'iChild'/'that'->_nbEntities and the amplitude of the mutation
// is equal to (max-min).(gauss(0.0, 1.0)+deltaAdn).ln('parents[0]'.age)
void GAMute(GenAlg* that, int* parents, 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 >= that->_nbEntities) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'child' is invalid (0<=%d<%d)",
      iChild, that->_nbEntities);
    PBErrCatch(GenAlgErr);
  }
#endif
  // Get the first parent and child
  GenAlgEntity* parentA = GAEntity(that, parents[0]);
  GenAlgEntity* child = GAEntity(that, iChild);
  // Get the probability of mutation
  float probMute = ((float)iChild) / ((float)GAGetNbEntities(that));
  // Get the amplitude of mutation
  float amp = 1.0 - 1.0 / sqrt((float)(parentA->_age + 1));
  // For each gene of the adn for floating point value
  for (int iGene = GAGetLengthAdnFloat(that); iGene--;) {
    // If this gene mutes
    if (rnd() < probMute) {
      // Get the bounds
      VecFloat2D* bounds = GABoundsAdnFloat(that, iGene);
      // Declare a variable to memorize the previous value of the gene
      float prevVal = GAEntGetGeneF(child, iGene);
      // Apply the mutation
      GAEntSetGeneF(child, iGene, GAEntGetGeneF(child, iGene) +
        (VecGet(bounds, 1) - VecGet(bounds, 0)) * amp *
        (rnd() - 0.5 + GAEntGetDeltaGeneF(child, iGene)));
      // Keep the gene value in bounds
      while (GAEntGetGeneF(child, iGene) < VecGet(bounds, 0) ||
        GAEntGetGeneF(child, iGene) > VecGet(bounds, 1)) {
        if (GAEntGetGeneF(child, iGene) > VecGet(bounds, 1))
          GAEntSetGeneF(child, iGene,
            2.0 * VecGet(bounds, 1) - GAEntGetGeneF(child, iGene));
        else if (GAEntGetGeneF(child, iGene) < VecGet(bounds, 0))
          GAEntSetGeneF(child, iGene,
```

```
            2.0 * VecGet(bounds, 0) - GAEntGetGeneF(child, iGene));
      }
      // Update the deltaAdn
      GAEntSetDeltaGeneF(child, iGene,
        GAEntGetGeneF(child, iGene) - prevVal);
    }
  }
  // For each gene of the adn for int value
  for (int iGene = GAGetLengthAdnInt(that); iGene--;) {
    // If this gene mutes
    if (rnd() < probMute) {
      // Get the bounds
      VecShort2D* boundsI = GABoundsAdnInt(that, iGene);
      VecFloat2D bounds = VecShortToFloat2D(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);
      GAEntSetGeneI(child, iGene, GAEntGetGeneI(child, iGene) +
        (short)round(ampI * (rnd() - 0.5)));
      // Keep the gene value in bounds
      while (GAEntGetGeneI(child, iGene) < VecGet(&bounds, 0) ||
        GAEntGetGeneI(child, iGene) > VecGet(&bounds, 1)) {
        if (GAEntGetGeneI(child, iGene) > VecGet(&bounds, 1))
          GAEntSetGeneI(child, iGene,
            2 * VecGet(&bounds, 1) - GAEntGetGeneI(child, iGene));
        else if (GAEntGetGeneI(child, iGene) < VecGet(&bounds, 0))
          GAEntSetGeneI(child, iGene,
            2 * VecGet(&bounds, 0) - GAEntGetGeneI(child, iGene));
      }
    }
  }
}

// Print the information about the GenAlg 'that' on the stream 'stream'
void GAPrintln(GenAlg* that, FILE* 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:%d - run:%d\n",
    GAGetCurEpoch(that), GAGetCurRun(that));
  fprintf(stream, "%d entities, %d elites\n", GAGetNbEntities(that),
    GAGetNbElites(that));
  for (int iEnt = 0; iEnt < GAGetNbEntities(that); ++iEnt) {
    GenAlgEntity* ent = GAEntity(that, iEnt);
    fprintf(stream, "#%d elo:%f ", iEnt,
      ELORankGetELO(GAEloRank(that), ent));
    if (iEnt < GAGetNbElites(that))
      fprintf(stream, "elite ");
    GAEntPrintln(ent, stream);
  }
}
```

```
// Get the level of inbreeding of curent entities of the GenAlg 'that'
// The return value is in [0.0, 1.0]
// 0.0 means all the elite entities have exactly the same adns
float GAGetInbreeding(GenAlg* that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  // Declare a variable to memorize the result
  float inbreeding = 0.0;
  // Declare a variable for calculation
  int nb = 1;
  // If there are adn for floating point values
  if (GAGetLengthAdnFloat(that) > 0) {
    // Declare a vector to memorize the ranges in gene values
    VecFloat* range = VecFloatCreate(GAGetLengthAdnFloat(that));
    // Calculate the ranges in gene values
    for (int iGene = GAGetLengthAdnFloat(that); iGene--;)
      VecSet(range, iGene,
        VecGet(GABoundsAdnFloat(that, iGene), 1) -
        VecGet(GABoundsAdnFloat(that, iGene), 0));
    // Calculate the norm of the range
    float normRange = VecNorm(range);
    // For each elite entity except the first one
    for (int iEnt = 1; iEnt < GAGetNbElites(that); ++iEnt) {
      // Get the difference in adn with the first entity
      VecFloat* diff = VecGetOp(GAEntAdnF(GAEntity(that, iEnt)), 1.0,
        GAEntAdnF(GAEntity(that, 0)), -1.0);
      // Calculate the inbreeding
      inbreeding += VecNorm(diff) / normRange;
      // Free memory
      VecFree(&diff);
    }
    // Calculate the inbreeding
    nb += GAGetNbElites(that);
    // Free memory
    VecFree(&range);
  }
  // If there are adn for floating point values
  if (GAGetLengthAdnInt(that) > 0) {
    // Declare a vector to memorize the ranges in gene values
    VecFloat* range = VecFloatCreate(GAGetLengthAdnInt(that));
    // Calculate the ranges in gene values
    for (int iGene = GAGetLengthAdnInt(that); iGene--;)
      VecSet(range, iGene,
        (float)(VecGet(GABoundsAdnInt(that, iGene), 1) -
        VecGet(GABoundsAdnInt(that, iGene), 0)));
    // Calculate the norm of the range
    float normRange = VecNorm(range);
    // For each elite entity except the first one
    for (int iEnt = 1; iEnt < GAGetNbElites(that); ++iEnt) {
      // Get the difference in adn with the first entity
      VecShort* diff = VecGetOp(GAEntAdnI(GAEntity(that, iEnt)), 1,
        GAEntAdnI(GAEntity(that, 0)), -1);
      VecFloat* diffF = VecShortToFloat(diff);
      // Calculate the inbreeding
      inbreeding += VecNorm(diffF) / normRange;
      // Free memory
      VecFree(&diffF);
```

```
      VecFree(&diff);
    }
    // Calculate the inbreeding
    nb += GAGetNbElites(that);
    // Free memory
    VecFree(&range);
  }
  // Calculate the inbreeding
  inbreeding /= (float)nb;
  // Return the result
  return inbreeding;
}

// 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* 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
  // If 'that' is already allocated
  if (*that != NULL) {
    // Free memory
    GenAlgFree(that);
  }
  // Load the number of entity and elite, and the length of adn
  int nbEnt, nbElite, lenAdnF, lenAdnI;
  int ret = fscanf(stream, "%d %d %d %d", &nbEnt, &nbElite,
    &lenAdnF, &lenAdnI);
  // If we couldn't fscanf
  if (ret == EOF)
    return false;
  // Check the data
  if (nbEnt < 3 || nbElite < 2 || lenAdnF < 0 || lenAdnI < 0)
    return false;
  // Allocate memory
  *that = GenAlgCreate(nbEnt, nbElite, lenAdnF, lenAdnI);
  // Load the run, epoch, nbRunPerEpoch, nextId
  ret = fscanf(stream, "%d %d %d %d", &((*that)->_curRun),
    &((*that)->_curEpoch), &((*that)->_runsPerEpoch),
    &((*that)->_nextId));
  // If we couldn't fscanf
  if (ret == EOF)
    return false;
  // Load the bounds
  for (int iBound = 0; iBound < lenAdnF; ++iBound) {
    VecFloat* b = NULL;
    if (VecLoad(&b, stream) == false)
      return false;
    VecCopy(GABoundsAdnFloat(*that, iBound), b);
    VecFree(&b);
  }
  for (int iBound = 0; iBound < lenAdnI; ++iBound) {
```

```
    VecShort* b = NULL;
    if (VecLoad(&b, stream) == false)
      return false;
    VecCopy(GABoundsAdnInt(*that, iBound), b);
    VecFree(&b);
  }
  // Load the ELO rank
  for (int iEnt = 0; iEnt < nbEnt; ++iEnt) {
    GSetElem* setElem = GSetGetElem(&(GAEloRank(*that)->_set), iEnt);
    ELOEntity* eloEnt = (ELOEntity*)(setElem->_data);
    GenAlgEntity* ent = (GenAlgEntity*)(eloEnt->_data);
    // Load the id, age and elo
    int id, age;
    float elo;
    int ret = fscanf(stream, "%d %d %f", &id, &age, &elo);
    // If we couldn't fscanf
    if (ret == EOF)
      return false;
    // Set the id and elo
    ent->_id = id;
    ent->_age = age;
    setElem->_sortVal = elo;
    // Load the genes
    if (lenAdnF > 0) {
      if (VecLoad(&(ent->_adnF), stream) == false)
        return false;
      if (VecLoad(&(ent->_deltaAdnF), stream) == false)
        return false;
    }
    if (lenAdnI > 0)
      if (VecLoad(&(ent->_adnI), stream) == false)
        return false;
  }
  // Return success code
  return true;
}

// Save the GenAlg 'that' to the stream 'stream'
// Return true in case of success, else false
bool GASave(GenAlg* that, FILE* 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
  // Save the number of entity and elite, and the length of adn
  int ret = fprintf(stream, "%d %d %d %d\n", GAGetNbEntities(that),
    GAGetNbElites(that), GAGetLengthAdnFloat(that),
    GAGetLengthAdnInt(that));
  // If we couldn't fprintf
  if (ret < 0)
    return false;
  // Save the run, epoch, nbRunPerEpoch, nextId
  ret = fprintf(stream, "%d %d %d %d\n", GAGetCurRun(that),
    GAGetCurEpoch(that), GAGetRunsPerEpoch(that), that->_nextId);
```

```
  // If we couldn't fprintf
  if (ret < 0)
    return false;
  // Save the bounds
  for (int iBound = 0; iBound < GAGetLengthAdnFloat(that); ++iBound)
    if (VecSave(GABoundsAdnFloat(that, iBound), stream) == false)
      return false;
  for (int iBound = 0; iBound < GAGetLengthAdnInt(that); ++iBound)
    if (VecSave(GABoundsAdnInt(that, iBound), stream) == false)
      return false;
  // Save the ELO rank
  for (int iEnt = 0; iEnt < GAGetNbEntities(that); ++iEnt) {
    GSetElem* setElem = GSetGetElem(&(GAEloRank(that)->_set), iEnt);
    ELOEntity* eloEnt = (ELOEntity*)(setElem->_data);
    GenAlgEntity* ent = (GenAlgEntity*)(eloEnt->_data);
    // Save the id, age and elo
    int ret = fprintf(stream, "%d %d %f\n", ent->_id, ent->_age,
      setElem->_sortVal);
    // If we couldn't fprintf
    if (ret < 0)
      return false;
    // Save the genes
    if (GAGetLengthAdnFloat(that) > 0) {
      if (VecSave(ent->_adnF, stream) == false)
        return false;
      if (VecSave(ent->_deltaAdnF, stream) == false)
        return false;
    }
    if (GAGetLengthAdnInt(that) > 0)
      if (VecSave(ent->_adnI, stream) == false)
        return false;
  }
  // Return success code
  return true;
}
```

3.2 genalg-inline.c

```
// ============ GENALG-INLINE.C ================

// ------------- GenAlgEntity

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

// Return the adn for floating point values of the GenAlgEntity 'that'
#if BUILDMODE != 0
inline
#endif
VecFloat* GAEntAdnF(GenAlgEntity* 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
// GenAlgEntity 'that'
#if BUILDMODE != 0
inline
#endif
VecFloat* GAEntDeltaAdnF(GenAlgEntity* 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 GenAlgEntity 'that'
#if BUILDMODE != 0
inline
#endif
VecShort* GAEntAdnI(GenAlgEntity* 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
// GenAlgEntity 'that'
#if BUILDMODE != 0
inline
#endif
float GAEntGetGeneF(GenAlgEntity* that, int 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 GenAlgEntity 'that'
#if BUILDMODE != 0
inline
#endif
float GAEntGetDeltaGeneF(GenAlgEntity* that, int 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
// GenAlgEntity 'that'
#if BUILDMODE != 0
inline
#endif
int GAEntGetGeneI(GenAlgEntity* that, int 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
// GenAlgEntity 'that' to 'gene'
#if BUILDMODE != 0
inline
#endif
void GAEntSetGeneF(GenAlgEntity* that, int iGene, float gene) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  VecSet(that->_adnF, iGene, gene);
}

// Set the delta of the 'iGene'-th gene of the adn for floating point
// values of the GenAlgEntity 'that' to 'delta'
#if BUILDMODE != 0
inline
#endif
void GAEntSetDeltaGeneF(GenAlgEntity* that, int iGene, float delta) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  VecSet(that->_deltaAdnF, iGene, delta);
}

// Set the 'iGene'-th gene of the adn for int values of the
// GenAlgEntity 'that'to 'gene'
#if BUILDMODE != 0
inline
#endif
void GAEntSetGeneI(GenAlgEntity* that, int iGene, short gene) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
```

```
  VecSet(that->_adnI, iGene, gene);
}

// Get the id of the GenAlgEntity 'that'
#if BUILDMODE != 0
inline
#endif
int GAEntGetId(GenAlgEntity* that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_id;
}

// Get the age of the GenAlgEntity 'that'
#if BUILDMODE != 0
inline
#endif
int GAEntGetAge(GenAlgEntity* that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_age;
}

// ------------- GenAlg

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

// Return the ELORank of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
ELORank* GAEloRank(GenAlg* that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_elo;
}

// Return the nb of runs per epoch of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
int GAGetRunsPerEpoch(GenAlg* that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
```

```
  }
#endif
  return that->_runsPerEpoch;
}

// Return the nb of entities of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
int GAGetNbEntities(GenAlg* that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_nbEntities;
}

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

// Return the current run of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
int GAGetCurRun(GenAlg* that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_curRun;
}

// Return the current epoch of the GenAlg 'that'
#if BUILDMODE != 0
inline
#endif
int GAGetCurEpoch(GenAlg* that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
```

```
  return that->_curEpoch;
}

// Set the nb of runs per epoch of the GenAlg 'that' to 'runs'
// 'runs' must be greater than 0
#if BUILDMODE != 0
inline
#endif
void GASetRunsPerEpoch(GenAlg* that, int runs) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (runs <= 0) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'runs' is invalid (%d>0)", runs);
    PBErrCatch(GenAlgErr);
  }
#endif
  that->_runsPerEpoch = runs;
}

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

// Get the length of adn for integer value
#if BUILDMODE != 0
inline
#endif
int GAGetLengthAdnInt(GenAlg* that) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_lengthAdnI;
}

// Get the bounds for the 'iGene'-th gene of adn for floating point
// values
#if BUILDMODE != 0
inline
#endif
VecFloat2D* GABoundsAdnFloat(GenAlg* that, int iGene) {
#if BUILDMODE == 0
  if (that == NULL) {
```

```
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (iGene < 0 || iGene >= that->_lengthAdnF) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'iGene' is invalid (0<=%d<%d)",
      iGene, that->_lengthAdnF);
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_boundsF + iGene;
}

// Get the bounds for the 'iGene'-th gene of adn for integer values
#if BUILDMODE != 0
inline
#endif
VecShort2D* GABoundsAdnInt(GenAlg* that, int iGene) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (iGene < 0 || iGene >= that->_lengthAdnI) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'iGene' is invalid (0<=%d<%d)",
      iGene, that->_lengthAdnI);
    PBErrCatch(GenAlgErr);
  }
#endif
  return that->_boundsI + iGene;
}

// Get the GenAlgEntity of the GenAlg 'that' currently at rank 'iRank'
#if BUILDMODE != 0
inline
#endif
GenAlgEntity* GAEntity(GenAlg* that, int iRank) {
#if BUILDMODE == 0
  if (that == NULL) {
    GenAlgErr->_type = PBErrTypeNullPointer;
    sprintf(GenAlgErr->_msg, "'that' is null");
    PBErrCatch(GenAlgErr);
  }
  if (iRank < 0 || iRank >= that->_nbEntities) {
    GenAlgErr->_type = PBErrTypeInvalidArg;
    sprintf(GenAlgErr->_msg, "'iRank' is invalid (0<=%d<%d)",
      iRank, that->_nbEntities);
    PBErrCatch(GenAlgErr);
  }
#endif
  return (GenAlgEntity*)(ELORankGetRanked(that->_elo, iRank)->_data);
}
```

4 Makefile

```
#directory
PBERRDIR=../PBErr
PBMATHDIR=../PBMath
GSETDIR=../GSet
ELORANKDIR=../ELORank

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

include $(PBERRDIR)/Makefile.inc

INCPATH=-I./ -I$(PBERRDIR)/ -I$(GSETDIR)/ -I$(PBMATHDIR)/ -I$(ELORANKDIR)/
BUILDOPTIONS=$(BUILDPARAM) $(INCPATH)

# compiler
COMPILER=gcc

#rules
all : main

main: main.o pberr.o gset.o elorank.o pbmath.o genalg.o genalg.o Makefile
$(COMPILER) main.o pberr.o gset.o elorank.o pbmath.o genalg.o $(LINKOPTIONS) -o main

main.o : main.c $(PBERRDIR)/pberr.h $(GSETDIR)/gset.h $(ELORANKDIR)/elorank.h genalg.h genalg-inline.c Makefile
$(COMPILER) $(BUILDOPTIONS) -c main.c

genalg.o : genalg.c genalg.h genalg-inline.c Makefile
$(COMPILER) $(BUILDOPTIONS) -c genalg.c

elorank.o : $(ELORANKDIR)/elorank.c $(ELORANKDIR)/elorank.h $(ELORANKDIR)/elorank-inline.c Makefile
$(COMPILER) $(BUILDOPTIONS) -c $(ELORANKDIR)/elorank.c

pberr.o : $(PBERRDIR)/pberr.c $(PBERRDIR)/pberr.h Makefile
$(COMPILER) $(BUILDOPTIONS) -c $(PBERRDIR)/pberr.c

pbmath.o : $(PBMATHDIR)/pbmath.c $(PBMATHDIR)/pbmath-inline.c $(PBMATHDIR)/pbmath.h Makefile $(PBERRDIR)/pberr.h
$(COMPILER) $(BUILDOPTIONS) -c $(PBMATHDIR)/pbmath.c

gset.o : $(GSETDIR)/gset.c $(GSETDIR)/gset-inline.c $(GSETDIR)/gset.h Makefile $(PBERRDIR)/pberr.h
$(COMPILER) $(BUILDOPTIONS) -c $(GSETDIR)/gset.c

clean :
rm -rf *.o main

valgrind :
valgrind -v --track-origins=yes --leak-check=full --gen-suppressions=yes --show-leak-kinds=all ./main

unitTest :
main > unitTest.txt; diff unitTest.txt unitTestRef.txt
```

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 UnitTestGenAlgEntityCreateFree() {
  int id = 1;
  int lengthAdnF = 2;
  int lengthAdnI = 3;
  GenAlgEntity* ent = GenAlgEntityCreate(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, "GenAlgEntityCreate failed");
    PBErrCatch(GenAlgErr);
  }
  GenAlgEntityFree(&ent);
  if (ent != NULL) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GenAlgEntityFree failed");
    PBErrCatch(GenAlgErr);
  }
  printf("UnitTestGenAlgEntityCreateFree OK\n");
}

void UnitTestGenAlgEntityGetSet() {
  int id = 1;
  int lengthAdnF = 2;
  int lengthAdnI = 3;
  GenAlgEntity* ent = GenAlgEntityCreate(id, lengthAdnF, lengthAdnI);
  if (GAEntAdnF(ent) != ent->_adnF) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAEntAdnF failed");
    PBErrCatch(GenAlgErr);
  }
  if (GAEntDeltaAdnF(ent) != ent->_deltaAdnF) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAEntDeltaAdnF failed");
    PBErrCatch(GenAlgErr);
  }
  if (GAEntAdnI(ent) != ent->_adnI) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAEntAdnI failed");
    PBErrCatch(GenAlgErr);
  }
  GAEntSetGeneF(ent, 0, 1.0);
  if (ISEQUALF(VecGet(ent->_adnF, 0), 1.0) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAEntSetGeneF failed");
    PBErrCatch(GenAlgErr);
  }
  if (ISEQUALF(GAEntGetGeneF(ent, 0), 1.0) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAEntGetGeneF failed");
    PBErrCatch(GenAlgErr);
```

```
  }
  GAEntSetDeltaGeneF(ent, 0, 2.0);
  if (ISEQUALF(VecGet(ent->_deltaAdnF, 0), 2.0) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAEntSetDeltaGeneF failed");
    PBErrCatch(GenAlgErr);
  }
  if (ISEQUALF(GAEntGetDeltaGeneF(ent, 0), 2.0) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAEntGetDeltaGeneF failed");
    PBErrCatch(GenAlgErr);
  }
  GAEntSetGeneI(ent, 0, 3);
  if (VecGet(ent->_adnI, 0) != 3) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAEntSetGeneI failed");
    PBErrCatch(GenAlgErr);
  }
  if (GAEntGetGeneI(ent, 0) != 3) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAEntGetGeneI failed");
    PBErrCatch(GenAlgErr);
  }
  if (GAEntGetAge(ent) != 1) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAEntGetAge failed");
    PBErrCatch(GenAlgErr);
  }
  if (GAEntGetId(ent) != id) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAEntGetId failed");
    PBErrCatch(GenAlgErr);
  }
  GenAlgEntityFree(&ent);
  printf("UnitTestGenAlgEntityGetSet OK\n");
}

void UnitTestGenAlgEntityInit() {
  srandom(5);
  int id = 1;
  int lengthAdnF = 2;
  int lengthAdnI = 2;
  GenAlgEntity* ent = GenAlgEntityCreate(id, lengthAdnF, lengthAdnI);
  GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
    lengthAdnF, lengthAdnI);
  VecFloat2D boundsF = VecFloatCreateStatic2D();
  VecShort2D boundsI = VecShortCreateStatic2D();
  VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
  VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
  VecCopy(GABoundsAdnFloat(ga, 0), &boundsF);
  VecCopy(GABoundsAdnFloat(ga, 1), &boundsF);
  VecCopy(GABoundsAdnInt(ga, 0), &boundsI);
  VecCopy(GABoundsAdnInt(ga, 1), &boundsI);
  GAEntInit(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, "GAEntInit failed");
    PBErrCatch(GenAlgErr);
  }
```

```
  GenAlgFree(&ga);
  GenAlgEntityFree(&ent);
  printf("UnitTestGenAlgEntityInit OK\n");
}

void UnitTestGenAlgEntity() {
  UnitTestGenAlgEntityCreateFree();
  UnitTestGenAlgEntityGetSet();
  UnitTestGenAlgEntityInit();
  printf("UnitTestGenAlgEntity OK\n");
}

void UnitTestGenAlgCreateFree() {
  int lengthAdnF = 2;
  int lengthAdnI = 3;
  GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
    lengthAdnF, lengthAdnI);
  if (ga->_runsPerEpoch != GENALG_RUNPEREPOCH ||
    ga->_curRun != 0 ||
    ga->_curEpoch != 0 ||
    ga->_nextId != GENALG_NBENTITIES ||
    ga->_nbEntities != GENALG_NBENTITIES ||
    ga->_nbElites != GENALG_NBELITES ||
    ga->_lengthAdnF != lengthAdnF ||
    ga->_lengthAdnI != lengthAdnI ||
    ELORankGetNb(GAEloRank(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 (GAEloRank(ga) != ga->_elo) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAEloRank failed");
    PBErrCatch(GenAlgErr);
  }
  if (GAGetRunsPerEpoch(ga) != GENALG_RUNPEREPOCH) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetRunsPerEpoch failed");
    PBErrCatch(GenAlgErr);
  }
  if (GAGetNbEntities(ga) != GENALG_NBENTITIES) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetNbEntities failed");
    PBErrCatch(GenAlgErr);
  }
  if (GAGetNbElites(ga) != GENALG_NBELITES) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetNbElites failed");
```

```
    PBErrCatch(GenAlgErr);
  }
  if (GAGetCurRun(ga) != 0) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetCurRun failed");
    PBErrCatch(GenAlgErr);
  }
  if (GAGetCurEpoch(ga) != 0) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetCurEpoch failed");
    PBErrCatch(GenAlgErr);
  }
  GASetRunsPerEpoch(ga, 10);
  if (GAGetRunsPerEpoch(ga) != 10) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GASetRunsPerEpoch failed");
    PBErrCatch(GenAlgErr);
  }
  GASetNbEntities(ga, 10);
  if (GAGetNbEntities(ga) != 10 ||
    GAGetNbElites(ga) != 9 ||
    ELORankGetNb(GAEloRank(ga)) != 10) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GASetNbEntities failed");
    PBErrCatch(GenAlgErr);
  }
  GASetNbElites(ga, 20);
  if (GAGetNbEntities(ga) != 21 ||
    GAGetNbElites(ga) != 20 ||
    ELORankGetNb(GAEloRank(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);
  }
  if (GABoundsAdnInt(ga, 1) != ga->_boundsI + 1) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GABoundsAdnInt 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();
  VecShort2D boundsI = VecShortCreateStatic2D();
  VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
  VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
  VecCopy(GABoundsAdnFloat(ga, 0), &boundsF);
  VecCopy(GABoundsAdnFloat(ga, 1), &boundsF);
  VecCopy(GABoundsAdnInt(ga, 0), &boundsI);
  VecCopy(GABoundsAdnInt(ga, 1), &boundsI);
  GAInit(ga);
  GenAlgEntity* ent =
    (GenAlgEntity*)(
    ((ELOEntity*)(GAEloRank(ga)->_set._head->_data))->_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();
  VecShort2D boundsI = VecShortCreateStatic2D();
  VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
  VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
  VecCopy(GABoundsAdnFloat(ga, 0), &boundsF);
  VecCopy(GABoundsAdnFloat(ga, 1), &boundsF);
  VecCopy(GABoundsAdnInt(ga, 0), &boundsI);
  VecCopy(GABoundsAdnInt(ga, 1), &boundsI);
  GAInit(ga);
  GAPrintln(ga, stdout);
  GenAlgFree(&ga);
  printf("UnitTestGenAlgInit OK\n");
}

void UnitTestGenAlgStepRun() {
  srandom(5);
  int lengthAdnF = 2;
  int lengthAdnI = 2;
  GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
    lengthAdnF, lengthAdnI);
  VecFloat2D boundsF = VecFloatCreateStatic2D();
  VecShort2D boundsI = VecShortCreateStatic2D();
  VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
  VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
  VecCopy(GABoundsAdnFloat(ga, 0), &boundsF);
  VecCopy(GABoundsAdnFloat(ga, 1), &boundsF);
  VecCopy(GABoundsAdnInt(ga, 0), &boundsI);
  VecCopy(GABoundsAdnInt(ga, 1), &boundsI);
  GASetNbElites(ga, 2);
  GASetNbEntities(ga, 3);
  GAInit(ga);
```

```
  GSet* rank = GSetCreate();
  for (int i = 3; i--;)
    GSetAddSort(rank, GAEntity(ga, i), 3.0 - (float)i);
  GAStepRun(ga, rank);
  if (GAGetCurRun(ga) != 1) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAStepRun failed");
    PBErrCatch(GenAlgErr);
  }
  for (int i = 3; i--;) {
    if (ELORankGetRank(ga->_elo, GSetGet(rank, i)) != 2 - i ||
      ISEQUALF(ELORankGetELO(ga->_elo, GSetGet(rank, i)),
        92.0 + (float)i * 8.0) == false ||
      GAEntity(ga, i)->_age != 2) {
      GenAlgErr->_type = PBErrTypeUnitTestFailed;
      sprintf(GenAlgErr->_msg, "GAStepRun failed");
      PBErrCatch(GenAlgErr);
    }
  }
  GSetFree(&rank);
  GenAlgFree(&ga);
  printf("UnitTestGenAlgStepRun OK\n");
}

void UnitTestGenAlgGetInbreeding() {
  srandom(5);
  int lengthAdnF = 2;
  int lengthAdnI = 2;
  GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
    lengthAdnF, lengthAdnI);
  VecFloat2D boundsF = VecFloatCreateStatic2D();
  VecShort2D boundsI = VecShortCreateStatic2D();
  VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
  VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
  VecCopy(GABoundsAdnFloat(ga, 0), &boundsF);
  VecCopy(GABoundsAdnFloat(ga, 1), &boundsF);
  VecCopy(GABoundsAdnInt(ga, 0), &boundsI);
  VecCopy(GABoundsAdnInt(ga, 1), &boundsI);
  GASetNbElites(ga, 2);
  GASetNbEntities(ga, 3);
  GAInit(ga);
  if (ISEQUALF(GAGetInbreeding(ga), 0.182041) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetInbreeding failed");
    PBErrCatch(GenAlgErr);
  }
  VecCopy(GAEntity(ga, 1)->_adnF, GAEntity(ga, 0)->_adnF);
  VecCopy(GAEntity(ga, 1)->_adnI, GAEntity(ga, 0)->_adnI);
  if (ISEQUALF(GAGetInbreeding(ga), 0.0) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAGetInbreeding failed");
    PBErrCatch(GenAlgErr);
  }
  GenAlgFree(&ga);
  printf("UnitTestGenAlgGetInbreeding OK\n");
}

void UnitTestGenAlgStepEpoch() {
  srandom(5);
  int lengthAdnF = 2;
  int lengthAdnI = 2;
  GenAlg* ga = GenAlgCreate(3, 2, lengthAdnF, lengthAdnI);
```

```
VecFloat2D boundsF = VecFloatCreateStatic2D();
VecShort2D boundsI = VecShortCreateStatic2D();
VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
VecCopy(GABoundsAdnFloat(ga, 0), &boundsF);
VecCopy(GABoundsAdnFloat(ga, 1), &boundsF);
VecCopy(GABoundsAdnInt(ga, 0), &boundsI);
VecCopy(GABoundsAdnInt(ga, 1), &boundsI);
GAInit(ga);
GSet* rank = GSetCreate();
for (int i = 3; i--;)
  GSetAddSort(rank, GAEntity(ga, i), 3.0 - (float)i);
GAStepRun(ga, rank);
printf("Before StepEpoch:\n");
GAPrintln(ga, stdout);
GenAlgEntity* child = GAEntity(ga, 2);
GAStepEpoch(ga);
printf("After StepEpoch:\n");
GAPrintln(ga, stdout);
if (ga->_nextId != 4 || GAEntGetId(child) != 3 ||
  GAEntGetAge(child) != 1 ||
  ISEQUALF(GAEntGetGeneF(child, 0), 0.755265) == false ||
  ISEQUALF(GAEntGetGeneF(child, 1), -0.209552) == false ||
  ISEQUALF(GAEntGetDeltaGeneF(child, 0), -0.032739) == false ||
  ISEQUALF(GAEntGetDeltaGeneF(child, 1), -0.206048) == false ||
  GAEntGetGeneI(child, 0) != 4 ||
  GAEntGetGeneI(child, 1) != 1 ||
  GAEntity(ga, 1) != child ||
  GAEntGetAge(GAEntity(ga, 0)) != 2 ||
  GAEntGetAge(GAEntity(ga, 2)) != 2 ||
  GAEntGetId(GAEntity(ga, 0)) != 2 ||
  GAEntGetId(GAEntity(ga, 2)) != 1 ||
  ISEQUALF(ELORankGetELO(GAEloRank(ga), child), 100.0) == false ||
  ISEQUALF(ELORankGetELO(GAEloRank(ga), GAEntity(ga, 0)),
    108.0) == false ||
  ISEQUALF(ELORankGetELO(GAEloRank(ga), GAEntity(ga, 2)),
    100.0) == false) {
  GenAlgErr->_type = PBErrTypeUnitTestFailed;
  sprintf(GenAlgErr->_msg, "GAStepEpoch failed");
  PBErrCatch(GenAlgErr);
}
VecCopy(GAEntity(ga, 1)->_adnF, GAEntity(ga, 0)->_adnF);
VecCopy(GAEntity(ga, 1)->_adnI, GAEntity(ga, 0)->_adnI);
GAStepEpoch(ga);
printf("After StepEpoch with interbreeding:\n");
GAPrintln(ga, stdout);
if (ga->_nextId != 6 || GAEntGetId(child) != 4 ||
  GAEntGetAge(child) != 1 ||
  ISEQUALF(GAEntGetGeneF(child, 0), 0.788004) == false ||
  ISEQUALF(GAEntGetGeneF(child, 1), -0.003504) == false ||
  ISEQUALF(GAEntGetDeltaGeneF(child, 0), -0.032739) == false ||
  ISEQUALF(GAEntGetDeltaGeneF(child, 1), -0.206048) == false ||
  GAEntGetGeneI(child, 0) != 3 ||
  GAEntGetGeneI(child, 1) != 1 ||
  GAEntity(ga, 2) != child ||
  GAEntGetAge(GAEntity(ga, 0)) != 2 ||
  GAEntGetAge(GAEntity(ga, 1)) != 1 ||
  GAEntGetId(GAEntity(ga, 0)) != 2 ||
  GAEntGetId(GAEntity(ga, 1)) != 5 ||
  ISEQUALF(ELORankGetELO(GAEloRank(ga), child), 100.0) == false ||
  ISEQUALF(ELORankGetELO(GAEloRank(ga), GAEntity(ga, 0)),
    108.0) == false ||
```

```
    ISEQUALF(ELORankGetELO(GAEloRank(ga), GAEntity(ga, 1)),
      100.0) == false) {
    GenAlgErr->_type = PBErrTypeUnitTestFailed;
    sprintf(GenAlgErr->_msg, "GAStepEpoch failed");
    PBErrCatch(GenAlgErr);
  }
  GSetFree(&rank);
  GenAlgFree(&ga);
  printf("UnitTestGenAlgStepEpoch OK\n");
}

void UnitTestGenAlgLoadSave() {
  srandom(5);
  int lengthAdnF = 2;
  int lengthAdnI = 2;
  GenAlg* ga = GenAlgCreate(3, 2, lengthAdnF, lengthAdnI);
  VecFloat2D boundsF = VecFloatCreateStatic2D();
  VecShort2D boundsI = VecShortCreateStatic2D();
  VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
  VecSet(&boundsI, 0, 1); VecSet(&boundsI, 1, 10);
  VecCopy(GABoundsAdnFloat(ga, 0), &boundsF);
  VecCopy(GABoundsAdnFloat(ga, 1), &boundsF);
  VecCopy(GABoundsAdnInt(ga, 0), &boundsI);
  VecCopy(GABoundsAdnInt(ga, 1), &boundsI);
  GAInit(ga);
  GAStepEpoch(ga);
  GSet* rank = GSetCreate();
  for (int i = 3; i--;)
    GSetAddSort(rank, GAEntity(ga, i), 3.0 - (float)i);
  GAStepRun(ga, rank);
  FILE* stream = fopen("./UnitTestGenAlgLoadSave.txt", "w");
  if (GASave(ga, stream) == 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->_curRun != gaLoad->_curRun ||
    ga->_runsPerEpoch != gaLoad->_runsPerEpoch ||
    ga->_nbEntities != gaLoad->_nbEntities ||
    ga->_nbElites != gaLoad->_nbElites ||
    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 ||
    GAEntGetId(GAEntity(ga, 0)) != GAEntGetId(GAEntity(gaLoad, 0)) ||
    GAEntGetId(GAEntity(ga, 1)) != GAEntGetId(GAEntity(gaLoad, 1)) ||
    GAEntGetId(GAEntity(ga, 2)) != GAEntGetId(GAEntity(gaLoad, 2)) ||
    GAEntGetAge(GAEntity(ga, 0)) != GAEntGetAge(GAEntity(gaLoad, 0)) ||
    GAEntGetAge(GAEntity(ga, 1)) != GAEntGetAge(GAEntity(gaLoad, 1)) ||
```

```
    GAEntGetAge(GAEntity(ga, 2)) != GAEntGetAge(GAEntity(gaLoad, 2)) ||
    VecIsEqual(GAEntity(ga, 0)->_adnF,
      GAEntity(gaLoad, 0)->_adnF) == false ||
    VecIsEqual(GAEntity(ga, 0)->_deltaAdnF,
      GAEntity(gaLoad, 0)->_deltaAdnF) == false ||
    VecIsEqual(GAEntity(ga, 0)->_adnI,
      GAEntity(gaLoad, 0)->_adnI) == false ||
    VecIsEqual(GAEntity(ga, 1)->_adnF,
      GAEntity(gaLoad, 1)->_adnF) == false ||
    VecIsEqual(GAEntity(ga, 1)->_deltaAdnF,
      GAEntity(gaLoad, 1)->_deltaAdnF) == false ||
    VecIsEqual(GAEntity(ga, 1)->_adnI,
      GAEntity(gaLoad, 1)->_adnI) == false ||
    VecIsEqual(GAEntity(ga, 2)->_adnF,
      GAEntity(gaLoad, 2)->_adnF) == false ||
    VecIsEqual(GAEntity(ga, 2)->_deltaAdnF,
      GAEntity(gaLoad, 2)->_deltaAdnF) == false ||
    VecIsEqual(GAEntity(ga, 2)->_adnI,
      GAEntity(gaLoad, 2)->_adnI) == false ||
    ISEQUALF(ELORankGetELO(GAEloRank(ga), GAEntity(ga, 0)),
      ELORankGetELO(GAEloRank(gaLoad), GAEntity(gaLoad, 0))) == false ||
    ISEQUALF(ELORankGetELO(GAEloRank(ga), GAEntity(ga, 1)),
      ELORankGetELO(GAEloRank(gaLoad), GAEntity(gaLoad, 1))) == false ||
    ISEQUALF(ELORankGetELO(GAEloRank(ga), GAEntity(ga, 2)),
      ELORankGetELO(GAEloRank(gaLoad), GAEntity(gaLoad, 2))) == 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(VecFloat* adnF, VecShort* 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(5);
  int lengthAdnF = 4;
  int lengthAdnI = lengthAdnF;
  GenAlg* ga = GenAlgCreate(GENALG_NBENTITIES, GENALG_NBELITES,
    lengthAdnF, lengthAdnI);
  VecFloat2D boundsF = VecFloatCreateStatic2D();
  VecShort2D boundsI = VecShortCreateStatic2D();
  VecSet(&boundsF, 0, -1.0); VecSet(&boundsF, 1, 1.0);
```

```
  VecSet(&boundsI, 0, 0); VecSet(&boundsI, 1, 4);
  for (int i = lengthAdnF; i--;) {
    VecCopy(GABoundsAdnFloat(ga, i), &boundsF);
    VecCopy(GABoundsAdnInt(ga, i), &boundsI);
  }
  GAInit(ga);
  GASetRunsPerEpoch(ga, 1);
  GSet* rank = GSetCreate();
//float best = 1.0;
  do {
    for (int iEnt = GAGetNbEntities(ga); iEnt--;)
      GSetAddSort(rank, GAEntity(ga, iEnt),
        -1.0 * evaluate(GAEntAdnF(GAEntity(ga, iEnt)),
        GAEntAdnI(GAEntity(ga, iEnt))));
    GAStepRun(ga, rank);
    GSetFlush(rank);
/*float ev = evaluate(GAEntAdnF(GAEntity(ga, 0)),
GAEntAdnI(GAEntity(ga, 0)));
printf("%d %f\r",GAGetCurEpoch(ga), ev);
if (best > ev) {
  best = ev;
  printf("\n");
  GAEntPrintln(GAEntity(ga, 0), stdout);
}*/
  } while (GAGetCurEpoch(ga) < 20000 ||
    evaluate(GAEntAdnF(GAEntity(ga, 0)),
      GAEntAdnI(GAEntity(ga, 0))) < PBMATH_EPSILON);
  printf("target: -0.5*x^3 + 0.314*x^2 - 0.7777*x + 0.1\n");
  printf("approx: \n");
  GAEntPrintln(GAEntity(ga, 0), stdout);
  printf("error: %f\n", evaluate(GAEntAdnF(GAEntity(ga, 0)),
    GAEntAdnI(GAEntity(ga, 0))));
  GSetFree(&rank);
  GenAlgFree(&ga);
  printf("UnitTestGenAlgTest OK\n");
}

void UnitTestGenAlg() {
  UnitTestGenAlgCreateFree();
  UnitTestGenAlgGetSet();
  UnitTestGenAlgInit();
  UnitTestGenAlgPrint();
  UnitTestGenAlgStepRun();
  UnitTestGenAlgGetInbreeding();
  UnitTestGenAlgStepEpoch();
  UnitTestGenAlgLoadSave();
  UnitTestGenAlgTest();
  printf("UnitTestGenAlg OK\n");
}

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

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

6 Unit tests output

```
UnitTestGenAlgEntityCreateFree OK
UnitTestGenAlgEntityGetSet OK
UnitTestGenAlgEntityInit OK
UnitTestGenAlgEntity OK
UnitTestGenAlgCreateFree OK
UnitTestGenAlgGetSet OK
UnitTestGenAlgInit OK
epoch:0 - run:0
3 entities, 2 elites
#0 elo:100.000000 elite id:2 age:1
  adnF:<0.788,-0.004>
  deltaAdnF:<0.000,0.000>
  adnI:<3,1>
#1 elo:100.000000 elite id:1 age:1
  adnF:<-0.841,-0.705>
  deltaAdnF:<0.000,0.000>
  adnI:<5,4>
#2 elo:100.000000 id:0 age:1
  adnF:<-0.907,-0.451>
  deltaAdnF:<0.000,0.000>
  adnI:<2,10>
UnitTestGenAlgInit OK
UnitTestGenAlgStepRun OK
UnitTestGenAlgGetInbreeding OK
Before StepEpoch:
epoch:0 - run:1
3 entities, 2 elites
#0 elo:108.000000 elite id:2 age:2
  adnF:<0.788,-0.004>
  deltaAdnF:<0.000,0.000>
  adnI:<3,1>
#1 elo:100.000000 elite id:1 age:2
  adnF:<-0.841,-0.705>
  deltaAdnF:<0.000,0.000>
  adnI:<5,4>
#2 elo:92.000000 id:0 age:2
  adnF:<-0.907,-0.451>
  deltaAdnF:<0.000,0.000>
  adnI:<2,10>
After StepEpoch:
epoch:1 - run:0
3 entities, 2 elites
#0 elo:108.000000 elite id:2 age:2
  adnF:<0.788,-0.004>
  deltaAdnF:<0.000,0.000>
  adnI:<3,1>
#1 elo:100.000000 elite id:3 age:1
  adnF:<0.755,-0.210>
  deltaAdnF:<-0.033,-0.206>
  adnI:<4,1>
#2 elo:100.000000 id:1 age:2
  adnF:<-0.841,-0.705>
  deltaAdnF:<0.000,0.000>
  adnI:<5,4>
After StepEpoch with interbreeding:
epoch:2 - run:0
3 entities, 2 elites
#0 elo:108.000000 elite id:2 age:2
  adnF:<0.788,-0.004>
```

```
  deltaAdnF:<0.000,0.000>
  adnI:<3,1>
#1 elo:100.000000 elite id:5 age:1
  adnF:<0.788,-0.153>
  deltaAdnF:<-0.033,-0.150>
  adnI:<3,1>
#2 elo:100.000000 id:4 age:1
  adnF:<0.788,-0.004>
  deltaAdnF:<-0.033,-0.206>
  adnI:<3,1>
UnitTestGenAlgStepEpoch OK
UnitTestGenAlgLoadSave OK
target: -0.5*x^3 + 0.314*x^2 - 0.7777*x + 0.1
approx:
id:1445228 age:5388
  adnF:<0.314,-0.500,0.100,-0.777>
  deltaAdnF:<0.048,0.000,-0.057,-0.000>
  adnI:<2,3,0,1>
error: 0.000274
UnitTestGenAlgTest OK
UnitTestGenAlg OK
UnitTestAll OK
```

UnitTestGenAlgLoadSave.txt:

```
3 2 2 2
1 1 100 4
2 -1.000000 1.000000
2 -1.000000 1.000000
2 1 10
2 1 10
1 2 92.000000
2 -0.840711 -0.704622
2 0.000000 0.000000
2 5 4
2 2 100.000000
2 0.788004 -0.003504
2 0.000000 0.000000
2 3 1
3 2 108.000000
2 0.765316 -0.146294
2 -0.022688 -0.142790
2 4 1
```