

GenAlg

P. Baillehache

October 30, 2018

Contents

1	Definitions	1
1.1	Selection	2
1.2	Reproduction	2
1.3	Mutation	2
2	Interface	3
3	Code	3
3.1	genalg.c	3
3.2	genalg-inline.c	3
4	Makefile	3
5	Unit tests	3
6	Unit tests output	3

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

3 Code

3.1 genalg.c

3.2 genalg-inline.c

4 Makefile

5 Unit tests

6 Unit tests output

UnitTestGenAlgLoadSave.txt:

eval() of best genes over epoch:

