GM – Genetic Manipulator

Gianluigi Ferraris

Genetic manipulator manages the genetic evolution of a set of "chromosomes". The chromosomes represent the genotype of a function from which we derive the evolution of the fitness. The Swarm is based on the Environment-Rules-Agents framework (Terna 1999) (hereafter ERA). Each agent owns a chromosome and uses an instance of the class Interface to obtain the meaning of its chromosome. Usually the chromosome is stored in the interface object but it can also be stored in a dataWarehouse, i.e. a special object made to store one or several chromosomes owned by an agent. In this case the interface is used only to translate the information encoded in the chromosomes. The manipulation is performed by the ruleMaker, which can interact with both interfaces and agents.

The RuleMaker of GM is able to perform reproduction, crossover and mutation of a set of arrays of characters which represent the chromosomes. This program can be easily included in every Swarm model. The reproduction is based on sexual matching, i. e. a single array cannot reproduce itself alone: the chromosome of the child is obtained by crossing those of the parents, however, a child may inherit the whole genetic heritage of a single parent. Birth rate and death rate are the same so the population is constant.

The RuleMaker acts accordingly to the value specified for the parameter **evolutionFrequency**; the user can set three ranges of values: i) If evolutionFrequency is a positive integer, evolution takes place each "evolutionFrequency" time, i. e. ruleMaker operates only if the computation of the module between number of steps and evolutionFrequency gives null. ii) If evolutionFrequency is set to 0, RuleMaker does not operate. iii) If evolutionFrequency is set between 0 and 1, the value is used as the evolution probability and the decision to trigger the genome evolution is made in a stochastic way.

The parameter "turnoverRate" determines the number of individuals involved in each evolution run. To make room for the newborn others individuals will die. TurnoverRate = 1 means that no parents will survive: this is an extreme way; high values of turnover can block the convergence of the system.

The children are obtained by copying the chosen parents and then crossing the children themselves with the probability given by parameter "**crossoverRate**". Mutation, made by switching a '0' to '1' or vice versa, is performed accordingly with the parameter "**mutationRate**". Each child can have an initial value for fitness, depending on the parameter childrenFitness. The parameter can be set to: i) zero or a positive real value, or ii) a negative value between 0 and –1. In the former case the value of the parameter is used as fixed amount of fitness that will be given to each new individual, whereas in the latter case the fitness is computed as follows:

fitnessOfChildOne = fitnessOfParentOne * childrenFitness + fitnessOfParentTwo * (1 – childrenFitness)
fitnessOfChildTwo = fitnessOfParentTwo * childrenFitness + fitnessOfParentOne * (1 – childrenFitness)

Parents are chosen randomly, with the probability of being picked up based on the fitness. Since probability to be picked up for reproduction is proportional to the fitness, the probability to die is inversely proportional. The new individuals replace those selected to die.

GM is able to handle also negative fitness values: in each evolution cycle the minimum fitness value is found, the amount is subtracted from the fitness value of each individual before performing selections. In this way, the selection is performed on the difference between the highest fitness value and the lowest. A second advantage is that, the best are the results of the individuals, the biggest become the influence of little difference in the fitness; so the algorithm becomes able to obtain more precise results than those obtained by using the whole fitness amount. This process is innovative: usually the whole value of fitness is used. Users can decide between this new management and the usual one by setting to '1' or '0' the value of parameter "useDeltaFitness".

1 - How to build an instance of RuleMaker

Between createBegin and createEnd, ruleMaker has to be told some values. Each of them can be modified during the run without any problem for the program. The values are:

- Address of the list of individuals: [.. **setPopulationList**: (id) ..]
- Length of a chromosome: (1 < integer < max length) [.. **setLengthOfGenoma**: (int) ..]
- TurnoverRate: (0 < real < 1) [.. **setTurnoverRate**: (float) ..]

-	CrossoverRate:	(0 < real < 1)	[.. **setCrossoverRate**: (float) ..]
-	MutationRate:	(0 < real < 1)	[..**setMutationRate**: (float) ..]
-	EvolutionFrequency:	(0 < real < 1) or (1 <= integer < 3.4E+38)	[..**setEvolutionFrequency**: (float) ..]
-	ChildrenFitness:	(-1 <= real < 0) or (0 <= real < 3.4E+38)	[..**setChildrenFitness**: (float) ..]
-	UseDeltaFitness	(0 or 1)	[**..setUseDeltaFitness**: (float) ..]

In createEnd method the program creates its own working lists where the contents of the population list will be copied. In this way the ruleMaker is highly independent and parallel actions on the population are allowed. Do not try to drop individuals during the action of the ruleMaker: the ids of the individuals are stored in the working lists of the ruleMaker and it may try to send something to those ids, if the corresponding object has been dropped the program can produce an error.

Pay attention in setting values of the parameters, values out of the specified ranges can induce unpredictable behaviours of the ruleMaker.

2 - How to use the Rule Maker

To evolve the population the user has, simply, to send the selector "step" to the ruleMaker. The programs uses five internal methods:

- keepData: to acquire data and compute number of individuals going to be involved in turnover.
- selectParents to pick up individuals that are going to be copied.
- selectDyings to pick up individuals that are going to die.
- shuffle to shuffle a list
- reproduce to make new individuals from the copies of the parents and replace the old ones.

The objects included in the populationList have to be instances of the classes "**Interface**" or "**Agent**". The agents have to answer the selector [.. **getInterface**] supplying the id of their own interface. The interfaces has to answer the three selectors: (float) [.. **getFitness**], (char *) [.. **getGenoma**] and [.. **setGenoma**: (char *) …]. The first gives the amount of fitness of the individual, the second returns the address of the array of char where is stored the representation of chromosome, and the third receives the pointer to the new chromosome.

3 - Customisation

In addition of the files RuleMaker.h and RuleMaker.m we use also the file Macro.h where are macros to help the user in customising the program. Macros are:

- **MAXGL**: "MAXimum Genoma Length" is the maximum number of position that can be managed. No problems occur , even for performances, when big arrays are used. The unique matter may be storage allocation.
- **AGENT(A)** // switching "//" to "A" instructions to manage lists of agents instead of lists of interfaces are enabled. Be careful in modifying the "import" instruction in file ruleMaker.h In so doing the user allows the management of a third kind of object. Note, also, that if we are using lists of agents, each of them needs an interface and has to be able to give the address of that interface as answer to the selector "getInterface". The message will be sent by the ruleMaker during housekeeping, i. e. the initial computational phase when the working lists are made and the parameters are read.
- **SHUFFLE(A)** // switching "//" to "A" the user can choose to shuffle the parentList before matching: in this way the computation become less deterministic.

4 - Statistics

The ruleMaker compute the number of operations that have been performed since the start of the run. Four counters are used to store the results: i) evolutions (times the population has been evolved), ii) reproductions (number of reproductions performed, each of them involving two parents and “giving birth” to two children) , iii) crossovers (number of children that have been crossed), iv) mutations (number of single positions whose value has been switched from '0' to '1' or vice versa).

To obtain these data, the user can send to the ruleMaker the selectors: getEvolutions, getReproductions, getCrossovers, getMutations. To print them on the standard output she can send the selector printStatistics.

5 - Demo, test e Trace

The sample model is intended only to show the working of the ruleMaker. The interfaces act directly as agents. The evolution is requested to the ruleMaker by the modelSwarm. Each time, before the evolutionary step, we send to the interfaces the order of setting randomly a fitness value.

Accordingly with the value of the parameter "displayFrequency" the population is printed on the standard output. The probe of the observer can set displayFrequency. For each individual, the code prints: address, body (a full stop is inserted after each group of four position to help reading) and the fitness value. Finally, statistics are printed.

The probe-map of the model allows for modifications in the number of interfaces, i. e. individuals, the length of each chromosome, the turnover, crossover and mutation rates.

The action of the ruleMaker can be investigated in more details using macros in the file Trace.h. Each of them can be activated as described for the macro Agent in the third paragraph. If a trace is activated, the specific object will print data at the beginning and at the end of the run of each method.

At the beginning of the method, the object prints: the address of the object, the name of the class, the name of the selector and, if they exist, the related parameters. At the end the object prints: the address of the object, the name of the class, the selector and the returned values. The start trace is identified by the word "entry" the ending one by the word "exit".

The traces between zero and five are intended to trace specific objects, while TRACE6 supplies more details about working lists, point of crossover and single positions that are going to be mutated. In this way the user is able to verify all the operations performed by the ruleMaker.

The outputs are sent to the standard output, usually the display. Using trace, it is better to switch these data to a file (using: “GM > filename” o “GM –s > filename”), of course several traces can be activated at the same time.