Java Jenetics Library
Capabilities Presentation

Victor-Alexandru Padurean

November 2019

1 Introduction

"Jenetics is designed with a clear separation of the several concepts of the algorithm, e.g. Gene, Chromosome, Genotype, Phenotype, Population and fitness Function. Jenetics allows you to minimize and maximize the given fitness function without tweaking it. In contrast to other GA implementations, the library uses the concept of an evolution stream (EvolutionStream) for executing the evolution steps. Since the EvolutionStream implements the Java Stream interface, it works smoothly with the rest of the Java Stream API. " [5]

Jenetics is a Java library, providing Genetic Algorithm and Genetic Programming support. It is compatible with newer versions of Java as well, not being dependent at runtime on any other libraries except for the Java 8 runtime.

The library uses a classical approach to Genetic Algorithms implementation:

```
$Population_0 \leftarrow Population_{initial}$
$CheckFitness(Population_0)$
while $!finished$ do
    $Generation \leftarrow Generation + 1$
    $Survivors_{Generation} \leftarrow select_{survivors}(Population_{Generation-1})$
    $Offspring_{Generation} \leftarrow select_{offspring}(Population_{Generation-1})$
    $Offspring_{Generation} \leftarrow alter(Offspring_{Generation})$
    $Population_{Generation} \leftarrow filter_{[Generation_i \geq Generation_{max}]}(Survivors_{Generation}) + filter_{[Generation_i \geq Generation_{max}]}(Offspring_{Generation})$
    $CheckFitness(Population_{Generation})$
end while
```

2 Architecture

2.1 Domain Classes

The domain classes form the domain model for the Genetic Algorithm, being comprised of Gene, Chromosome, Genotype and Phenotype.

Gene Genes are the basic blocks, containing as “Allele” the piece of information required by the programmer. It contains methods for Gene creation and validation. There are default classes implementing Gene, which contain pieces of information like characters, bits or longs, but one may create their own classes implementing the Gene interface, according to different utility scenarios.

Chromosome Chromosomes are collections of Genes. They contain at least one Gene and contain methods for Chromosome creation and validation, either from a given sequence of Genes or randomly. Chromosomes can be created with variable length, raging from a minimum to a maximum value.

Genotype The Genotype is the “structural and immutable representative of an individual”[6]. It is the class the Engine is working with. It may contain multiple Chromosomes with ranging lengths, but with the same Gene type.

Phenotype The Phenotype is the true representation of an individual, consisting of a Genotype and a fitness function. It forms the environment for the Genotype, but it doesn’t change its structure.

2.2 Operation Classes

The Operation Classes are used for selecting the potentially useful individuals for the final solution and altering them for genetic diversity. They are comprised of Alterers and Selectors.

2.2.1 Alterers

The two possibilities for altering individuals are mutation and recombination (crossover).

Mutation Mutation is useful for maintaining diversity and exploring the evolution space. The **Mutator** selects a random genotype from the population, then selects a random chromosome from the genotype and finally a random gene from the chromosome to alter. There are also **Gaussian mutator**, which picks a new value based on the Gaussian distribution around the current value of a number gene, and **Swap mutator**, which changes the order of the genes in the chromosome.

Recombination Recombination creates new chromosomes by combining parts from two ore more parent chromosome. There can be one ore more crossover points. **Single-point crossover** takes two individuals, splits them at a random point and mixes the genetic information from a parent with the complementary information from the other, obtaining a new individual. This type of recombination is slow. **Multi-point crossover** splits the two individuals’ chromosomes into multiple points. **Partially-matched crossover** works like the Multi-point crossover, but it guarantees that each gene is found exactly once in each chromosome, working for combinatorial problems. **Uniform crossover** swaps the genes at given indexes with a given probability. The **Partial alterer** works with the whole population as a sequence of genotypes; the **Mean alterer** and **Line crossover** are used for numeric chromosomes, the first computing the mean of the two values, whereas the second picks a value from the line described by the two points defined by the information present in the two genes.

2.2.2 Selectors

Tournament selector Here, the best individual is chosen from a set of randomly chosen individuals, according to the fitness function.

Truncation selector Individuals are sorted according to their fitness and only n individuals are selected.

Monte Carlo selector Individuals are chosen randomly from the population. In general, a selector must perform better than the Monte Carlo selector.

Roulette-wheel selector This selector selects individuals with a probability computed directly proportional with their fitness.

Linear-rank selector Roulette-wheel selector represents individual according to their score given by the fitness function, whereas Linear-rank selector represents them according to their rank after being sorted by the fitness function.

Exponential-rank selector It works similarly to the Linear-rank selection, but instead of a linear function, it uses an exponential one.

Boltzmann selector The Boltzmann selector uses the Boltzmann distribution function for selecting individuals.

Stochastic-universal selector "Stochastic-universal selector is a method for selecting individuals according to some given probability in a way that minimizes the chance of fluctuations."[6]

Elite selector This ensures that a proportion of individuals from the current generation is transferred to the next generation, without being changed.

2.3 Engine Classes

This are the classes that actually perform the evolution. On the EvolutionStream one can define a termination predicate and than collect the EvolutionResult.

2.3.1 Fitness Function

The fitness function is one of the most important parts when modeling the problem and algorithm. It should return a Comparable result, which will be used by the Engine to compare individuals' fitness.

2.3.2 Engine

The evolution Engine controls how the evolution steps are executed. It creates EvolutionStreams. The Engine parameters which can be configured are:

- alterers
- clock - used for calculating execution durations
- constraint - controls phenotype validity
- executor
- fitnessFunction
- genotypeFactory - used for creating new individuals
- maximalPhenotypeAge
- offspringFraction
- offspringSelector - used for specifying offspring selector
- optimize - used for maximizing or minimizing the fitness funciton
- populationSize
- selector - used for specifying survivors and offspring selectors all at once
- survivorsSelector - used for specifying survivors selector
- mapper - used for specifying a mapper that transforms the EvolutionResult after each generation

2.3.3 EvolutionStream

The EvolutionStream controls the execution of the evolution process and it can be used to define the termination criteria and control the evolution results.

2.3.4 EvolutionResult

This collects each evolution step into an immutable class. There are three types of Factories that will return Collectors usable for EvolutionStreams:

- toBestEvolutionResult()
- toBestPhenotype()
- toBestGenotype()

2.3.5 EvolutionStatistics

"TheEvolutionStatisticsclass allows you to gather additional statisticalinformation from theEvolutionStream. This is especially useful during thedevelopment phase of the application, when you have to find the right parametriza-tion of the evolutionEngine."[6]

2.3.6 Evaluator

The Evaluator is used for evaluating the fitness values for a given population.

3 Example

An easy to understand example that can present how the Jenetics library and more generally, a Genetic Algorithm, works is obtaining a given message. That is, starting from random strings of characters with a maximum length, the algorithm should obtain the given message followed by white spaces.

3.1 Formulation

The information quanta used in this problem are characters. There is no need in implementing a custom Gene containing a character as an allele, because the library already contains such gene class and chromosome class. Consequently, it is easy for a newcomer to make his first Genetic Algorithm based program as only a Fitness function is needed. A well formulated Fitness function is most important, because it makes the difference between the average case and the worst case, the latter being just a backtracking search. The rest of the evolution finesse details can be easily experimented with using the Engine configuration. For the proposed problem, a simple Fitness function is enough: we only need to check for each position, that is a Gene from a Chromosome, if it has the allele equal to the letter placed in the correspondent position of the solution string. If it is not the case, the individual will be penalized. The pseudocode is presented below:

```
procedure CHECKFITNESS($chromosome, string$)
    $score \leftarrow 0$
    $whitespace \leftarrow$ ""
    for $i from 1 to chromosome.length$ do
        if $i \leq string.length$ then
            if $chromosome[i].allele = string[i]$ then
                $score \leftarrow score + 1$
            end if
        else
            if $chromosome[i].allele = whitespace$ then
                $score \leftarrow score - 2$
            end if
        end if
        end for
    return score
    end procedure
```

In Java, it is easier to create another helper function (Listing 1) which tells how alike two Strings are and then use that function in the fitness function, after building the String the Chromosome represents.

```java
private static Long similarity(String a, String b) {
  long sum = 0;
  for (int   = 0; i < a.length() && i < b.length(); i++)
    if (a.charAt(i) == b.charAt(i))
      sum += 10;
  if (a.length() < b.length()) {
    for (int i = a.length(); i < b.length(); i++)
      if (b.charAt(i) != ' ')
        sum -= 20;
  } else {
    for (int i = b.length(); i < a.length(); i++)
      if (a.charAt(i) != ' ')
        sum -= 20;
  }
  return sum;
}
```

Listing 1: Helper function

The fitness function should build the representing String by passing through all the genes of the sole chromosome present inside the genotype and build a string by joining the letters. Finally, the helper function will return the difference between the obtained String and the final String.

```java
private static Long checkFitnessGeneticsHello(Genotype<
    CharacterGene> gt) {
    CharacterChromosome lc = gt.getChromosome().as(
    CharacterChromosome.class);

    String message = lc.stream().map(x -> "" + x.getAllele()).
    collect(Collectors.joining());

    System.out.println(message);

    return similarity(prop, message);
  }
```

Listing 2: Fitness function

For starting the evolution process, a Genotype Factory is needed, which will produce valid Genotypes according to the internal works of the CharacterChromosome class. An Engine must be built, using the Engine Builder, which will receive as parameters the Fitness Function and the Genotype producing Factory. Then it is configured with the desired selectors and alterers. To gain insights about the evolution process, a Consumer made by the EvolutionStatistics must be created. Finally, the engine is started after the limit is set, we peek at the statistics and collect the best Genotype. The pseudocode looks like this:

```
$Factory \leftarrow makeFactory(Genotype[CharacterChromosome[30]])$
$Engine \leftarrow buildEngine(checkFitness, Factory)$
$SurvivorsSelector_{Engine} \leftarrow TournamentSelection(5)$
$OffspringSelector_{Engine} \leftarrow RouletteWheelSelection$
$Alterers_{Engine} \leftarrow Alterer(Mutator(0.3), MultiPointCrossover(0.4))$
$Statistics \leftarrow EvolutionStatistics(CharacterGene)$
$Limit \leftarrow FitnessThreshold(similarity(prop, prop))$
$Result \leftarrow stream(Engine, Limit, Statistics)$
```

Java seems a bit more natural in working with Streams:

```java
prop = "Hello World!";

Factory<Genotype<CharacterGene>> gtf = Genotype.of(
  CharacterChromosome.of(30));

Engine<CharacterGene, Long> engine = Engine.builder(GeneticsHello
  ::checkFitnessGeneticsHello, gtf)
      .survivorsSelector(new TournamentSelector<>(5)).
  offspringSelector(new RouletteWheelSelector<>())
      .alterers(new Mutator<>(0.3), new MultiPointCrossover
  <>(0.4)).build();

Consumer<? super EvolutionResult<CharacterGene, Long>> statistics
   = EvolutionStatistics.ofNumber();

Genotype<CharacterGene> result = engine.stream().limit(limit.
  byFitnessThreshold(similarity(prop, prop) - 1))
      .peek(statistics).collect(EvolutionResult.toBestGenotype())
  ;

System.out.println(result);
System.out.println(checkFitnessGeneticsHello(result));
System.out.println(statistics);
```

Listing 3: Evolution

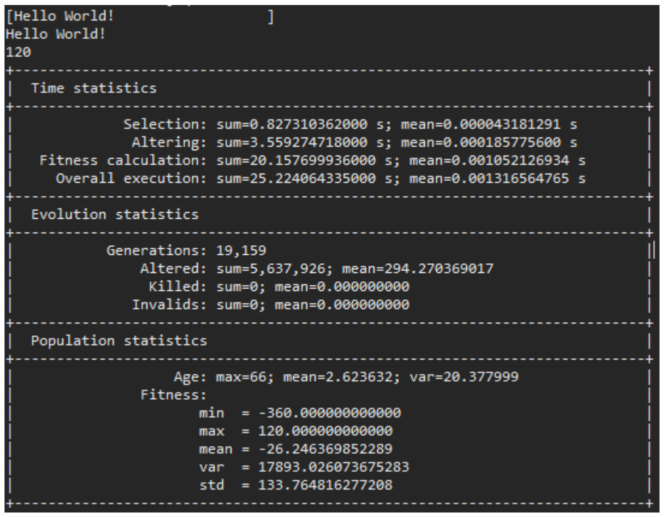
Configured like this, the Stream will not stop until the maximum score is achieved. The result and the statistics will be printed in the console. (Figure 1)

4 Applications

Genetic Algorithms have a wide variety of applications, raging from computer science to social sciences, from biological sciences to finance and economics.

4.1 Articles

Recurrent Neural Networks "Recurrent neural networks such as the fully interconnected Hopfield network have the potential of being applied to many in-



```
[Hello World!                 ]
Hello World!
120
+---------------------------------------------------------------------------+
|  Time statistics                                                          |
+---------------------------------------------------------------------------+
|             Selection: sum=0.827310362000 s; mean=0.000043181291 s        |
|              Altering: sum=3.559274718000 s; mean=0.000185775600 s        |
|   Fitness calculation: sum=20.157699936000 s; mean=0.001052126934 s       |
|     Overall execution: sum=25.224064335000 s; mean=0.001316564765 s       |
+---------------------------------------------------------------------------+
|  Evolution statistics                                                     |
+---------------------------------------------------------------------------+
|           Generations: 19,159                                             ||
|               Altered: sum=5,637,926; mean=294.270369017                  |
|                Killed: sum=0; mean=0.000000000                            |
|              Invalids: sum=0; mean=0.000000000                            |
+---------------------------------------------------------------------------+
|  Population statistics                                                    |
+---------------------------------------------------------------------------+
|                   Age: max=66; mean=2.623632; var=20.377999               |
|               Fitness:                                                    |
|                      min  = -360.000000000000                             |
|                      max  = 120.000000000000                              |
|                      mean = -26.246369852289                              |
|                      var  = 17893.026073675283                            |
|                      std  = 133.764816277208                              |
+---------------------------------------------------------------------------+
```

Figure 1: Results and Statistics

put output mapping problems, especially those requiring the outputs to change with time, such as a centeral pattern generator. However, due to the difficulty of training, such networks have not been as extensively utilized as the multilayer feedforward networks which can be trained using the back propagation algorithm. In this thesis we explore the applicability of the genetic algorithm as a search technique to find the network parameters for recurrent neural networks. We also investigate the potential of the genetic algorithm to determine not only the network parameters, but also the number of neurons to use in the network architecture. "[3]

Gene expression profiling analysis Gene expression profiling represents the measuring of the activity of thousand of genes at once, to create a global picture of their cellular funcion. "Identification of coordinately regulated genes according to the level of their expression during the time course of a process allows for discovering functional relationships among genes involved in the process."[4]

Design of anti-terrorism systems "Complex engineering systems are usually designed to last for many years. Such systems will face many uncertainties in the future. Hence the design and deployment of these systems should not be based on a single scenario, but should incorporate flexibility. Flexibility can be incorporated in system architectures in the form of options that can be exercised in the future when new information is available. Incorporating flexibility comes, however, at a cost. To evaluate if this cost is worth the investment a real options analysis can be carried out. This approach is demonstrated through analysis of a case study of a previously developed static system-of-systems for maritime domain protection in the Straits of Malacca. This article presents a framework for dynamic strategic planning of engineering systems using real options analysis and demonstrates that flexibility adds considerable value over a static design. In addition to this it is shown that Monte Carlo analysis and genetic algorithms can be successfully combined to find solutions in a case with a very large number of possible futures and system designs."[1]

Real options valuation Real option valuation applies option valuation techniques to capital budgeting decisions. "To address the issue of decision support for designing and managing flexible projects and systems in the face of uncertainties, this paper integrates real options valuation, decision analysis techniques, Monte Carlo simulations and evolutionary algorithms in an evolutionary real options framework. The proposed evolutionary real options framework searches for an optimized portfolio of real options and makes adaptive plans to cope with uncertainties as the future unfolds. Exemplified through a test case, the evolutionary framework not only compares favorably with traditional fixed design approaches but also delivers considerable improvements over prevailing real options practices."[2]

4.2 Personal Project

There is always a need for people to organize their time in order to be more productive and don't interfere with the plans of others. In the academic domain there is the classical timetable problem. It can be a real headache for the one managing the timetable and the ones having to comply as well. Genetic Algorithms may be applied to create a timetable that satisfy as many constraints as possible and not to fail if it cannot satisfy all of them. It should perform much better than a backtracking approach.

Problem formulation There are two sides in this problem: the students, organized in groups, and the teachers. For each one, two course should not take place at the same hour, in the same weekday. There can be additional constraints for groups, such as having at least three hours continuously, but not more than five hours continuously. Teacher may add their constraints too, for example one of them may have another job, so they would like to have one of the days free. The constraints should also be divided into to categories: hard and soft constraints. The algorithm should try to satisfy most of the constraints, but especially the hard ones.

References

[1] Joost Buurman; Stephen Zhang; Vladan Babovic. Reducing risk through real options in systems design: The case of architecting a maritime domain protection system.

[2] Stephen Zhang; Vladan Babovic. An evolutionary real options framework for the design and management of projects and systems with complex real options and exercising conditions.

[3] Omar Syed. Applying genetic algorithms to recurrent neural networks for learning network parameters and architecture.

[4] Cuong C To; Jiri Vohradsky. A parallel genetic algorithm for single class pattern classification and its application for gene expression profiling in streptomyces coelicolor.

[5] Franz Wilhelmstötter. Jenetics.

[6] Franz Wilhelmstötter. *Jenetics Library User's Manual 5.0.*