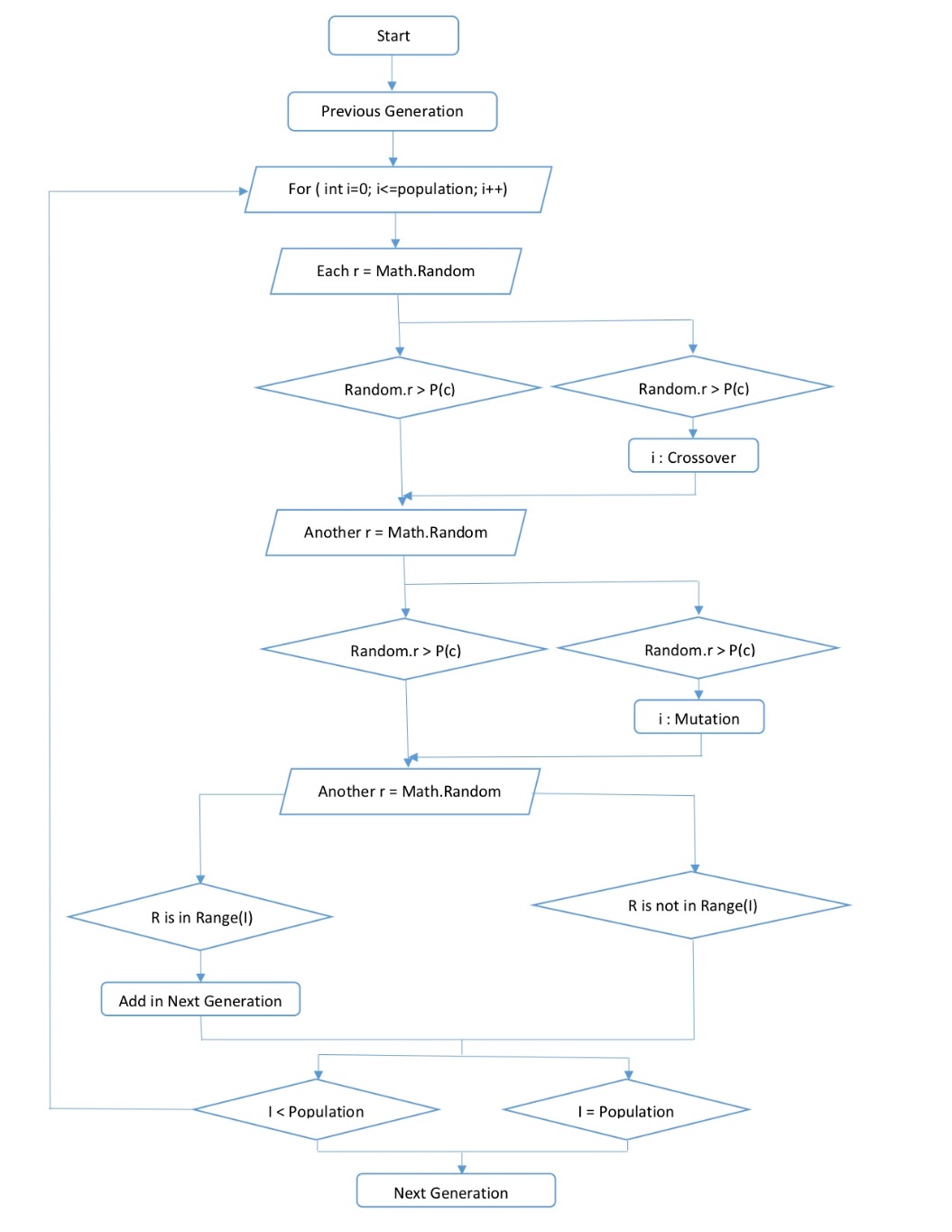
Genetic Algorithm Project Report

By Huai Huang, Hui Kai

We’ve solved two problems include TSP using GA. Another one is Polynomial Extrema Problem (PEP). The module is a generated polynomial function which contains up to 30 variables (could be more).

To solve this problem best, we first test the performance of GA on TSP, then on 2 variables polynomial, gradually increase the number of variables to 30. The main steps we use to build up genetic population is shown in the Flow Diagram below:

Specific parameters: included in config.properties.

Implementation: for TSP:

Genotype: an array of Integer, in which each number represents the number of city the salesman has been to (so this array has no duplicate key). The last index will be taken by the destination (which is the same as start). Total length equals to citiesNum+1;

Phenotype: The actual distance of a genotype;

Fitness: 1/d where d is the distance of the salesman takes;

For PEP:

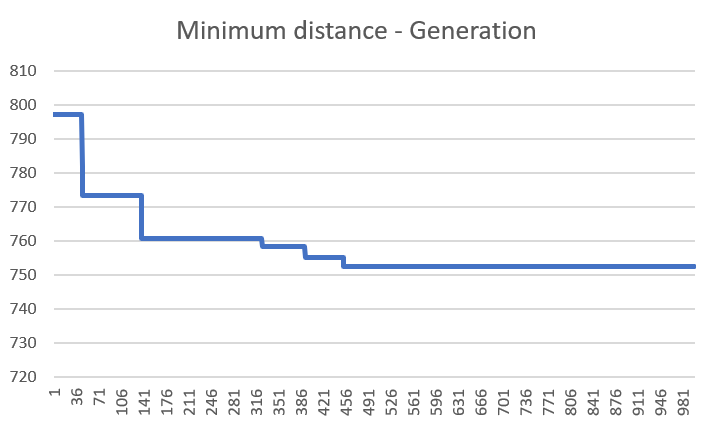
Genotype: an array of Double, in which each value is ranged from the minimum and maximum given in the configuration file; finally, we decide to use one to one function (one value represents one variable);

Phenotype: an array of Double, in which each value represents the actual value of each variable. The range of each value will be interpreted from the genotype, for example, -50~50 to -4~5. Always, range of genotype is larger than phenotype’s, which will make result more precisely;

Fitness: the fitness of each generation depends on the maximum function output (defined as max): fit = 1/(max - f(individual) + 10). Obviously, the more function value of an individual closed to the max, the greater of fit get;

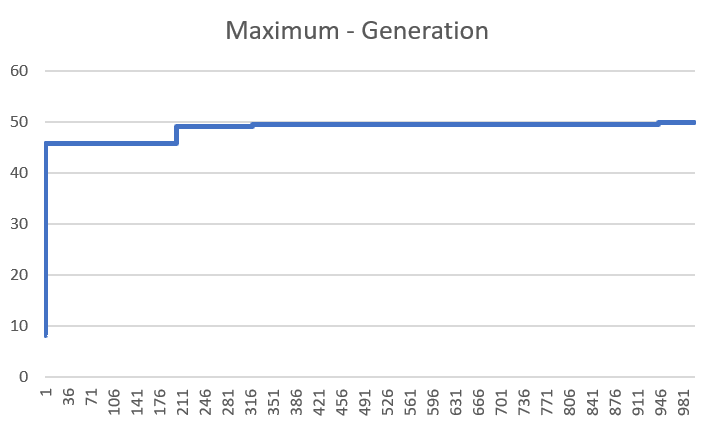
For both:

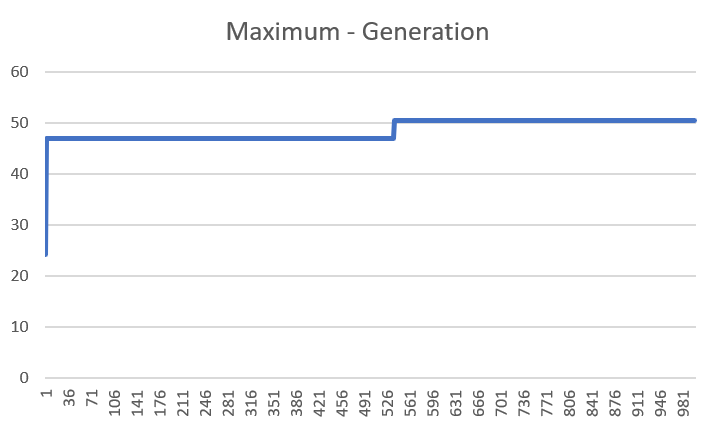
Selection: Roulette. Given an array of double in which a[i] (a.length = population+1) represents the accumulated value (obviously, a[0]=0 & a[last]=1), generate a random value r using Math.random(), if(r in (a[i], a[i+1])) then i is selected;

As we finished implementing these methods, we start to test the GA performance. Here is a result (maxGeneration = 1000):

It’s obviously that first 20% of breed descends fast, then the process slows down, gradually getting closed to the actual minimum. The whole process may accidentally exterminate by reaching single individual population (we don’t use the evolution function but only adjust the value of Pc-crossover probability and eliminated rate). No individual will “die” in this situation. We don’t consider it as a terrible condition in terms of the “roulette” selection. Finally, a tricky equation found: Pc ≈ (1-cutoff)/m in which cutoff represents eliminated rate and m represents mating generation. In this condition, cutoff = 0.5, if Pc = 0.25, population of each generation will be approximately the same.

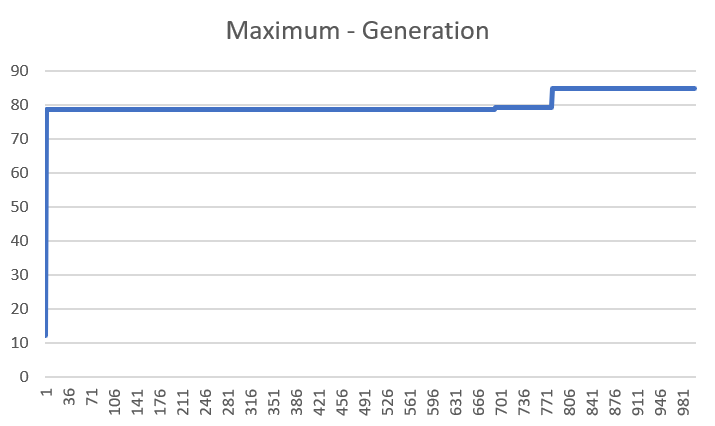
Then we work on the PEP.

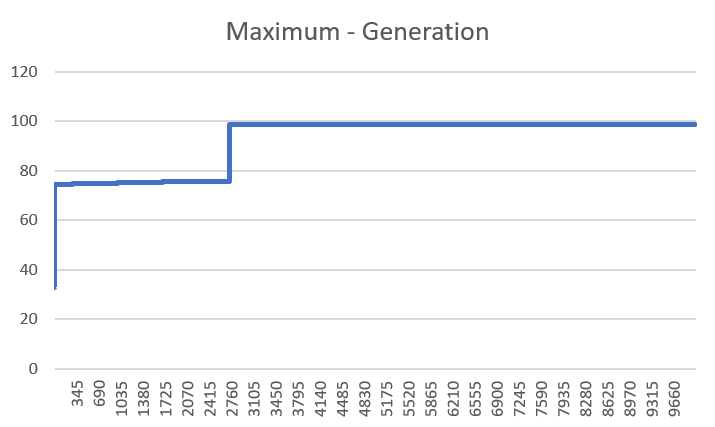
 Firstly, we use a simple function to see how GA works on PEP. In this function, given a range (-4, 5) to both x1 and x2, we can directly find out that the maximum value is 51 (when x1=x2=5). Let’s see how GA figures it out:

We can see that maximum rapidly get close to the boundary, but the actual value is 49.81, which is indeed close to 51 but not 51 (error = (51-49.81)/51 = 2.4%). During the test, we gradually find out that GA has a great difficulty to go “outbound” (which means get a new genotype out of the population). This difficulty maybe caused by the implementation of crossover: usually get a value **between** the parents. So, we change the implementation-it has a tiny probability to get out of its parents. Then we get a result like this:

Which the actual maximum is 50.514 and error=0.97% (much smaller).

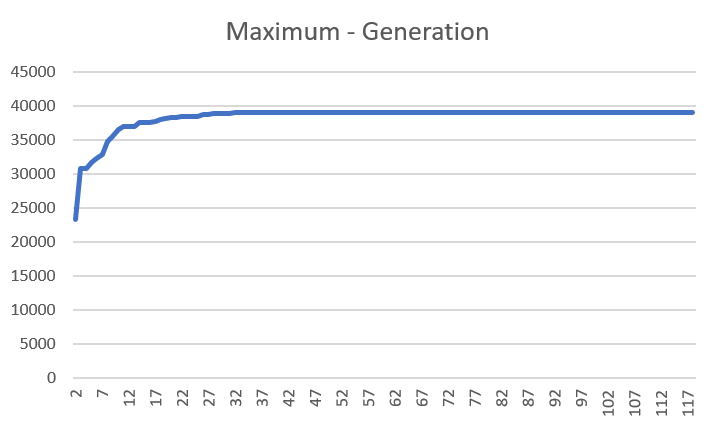
Secondly, we test on another function and maximum is obviously 101. But the result:



The maximum is 84.86, which is far from 101. We consider it was caused by the small number of generation. Then we change the max generation to 10000, aha, we face a severe problem: always terminated by single individual. We have to set a threshold to avoid this problem (due to there is no enough time to add evolution function), and add a special condition of mutation: symmetry. After these changes, we get a result like this:

The maximum is 98.74, Wow!

Finally, we get to the 30 variables problem. The function is generated in PolyGenoOperation.java. We first get a result shown below:



Maximum is 39055.96. The reason we only plot first 120 generation is that this value rarely grows after that (only grows 0.8 after 9880 generation). We can calculate the maximum of this function manually: it depends on the . We can assert that all the parameters of are negative, but on contrary, 4 of which are , their parameters are positive. So, the maximum is approximately 1.5\*10\*55 ≈ 40000. This result is very closed to the actual value.

In conclusion, we’ve figured out several points about GA during this project:

1. Crossover is the main aspect that affect the result;
2. Mutation can better solve problems which results are in boundaries;
3. GA Algorithm is suitable for many NP tasks with immutable variables length.