**Problem Description**

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

**Approach: Genetic Algorithm**

genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection. Considering our problem, we choose this algorithm as our solving approach.

**Our Implementation**

Considering the GAK class, it contains nine major methods: readData, initPopulation, evaluate, calcFitness, select, intersect, aberra, sort and solve. It also contains some other supplement methods like recBest methods.

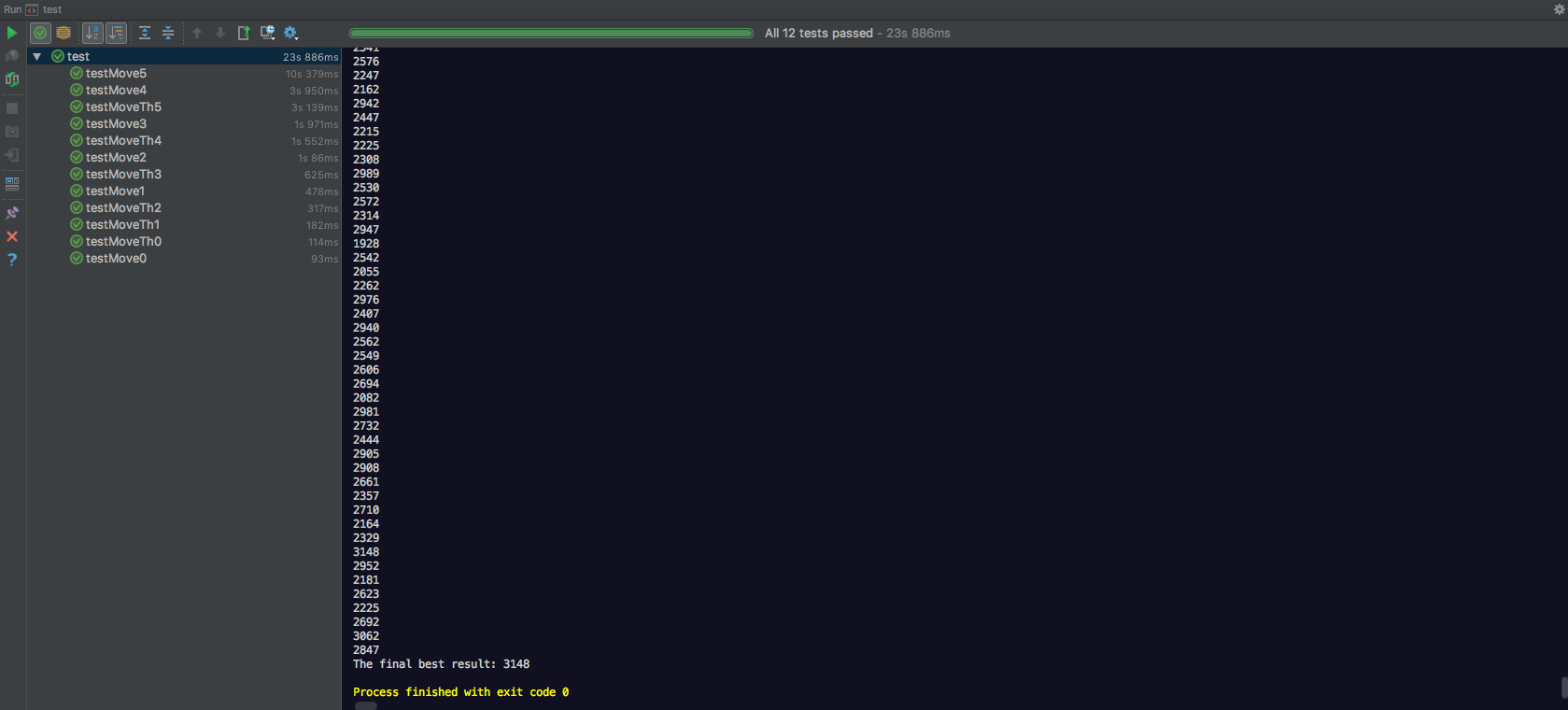
* The readData method reads the initial set for the weight and the value of each item that can be put into the bag.
* The initPolulation method initialize the process of solving Knapsack Problem, it builds fitness array and initialize population array with 1 for true, 0 for false. The true means that the item can be put into bag and false means the item cannot be put into the bag.
* The evaluate method calculate the weight and the value of the selecting strategy. As for each strategy of items selection to putting into bag, the method is used to calculate the total weight and the value of items that have been put into bag. If the weight of the items needs putting exceed the capacity of the bag, the method will return 0 to mean that this kind of strategy does not fit for the bag, otherwise it will return the total weight and the total value of items.
* The calcFitness method calculates the fitness of all individuals in the population.
* The select method is the selection strategy, which select the surviving individual from the roulette plate and randomly generating the remaining individuals.
* The intersect method receives the population from previous generation and then combine their order and generate the next generation. First, getting a random number between 0 and 1. Then comparing with the rate of crossover we set at first. If the random number is lower than the rate we set before, the population will get crossover, exchanging each other.
* The aberra method does mutation with the rate of mutation for individual and the rate of mutation which is the possibility of variation in each of the individuals that determines the mutation.
* Thesort method uses heap sort method to order the organisms by their fitness function.
* The solve method is the main method of the genetic algorithm, it combines the above methods and reproduce the next generation.

As for th class, we use a parallel computation mechanism so that we can divide the population up into sub-populations and create the next generations for each sub-population in parallel. The method contains the methods we have talked before in GAK class and the static CompletableFuture class to implement the parallel running.

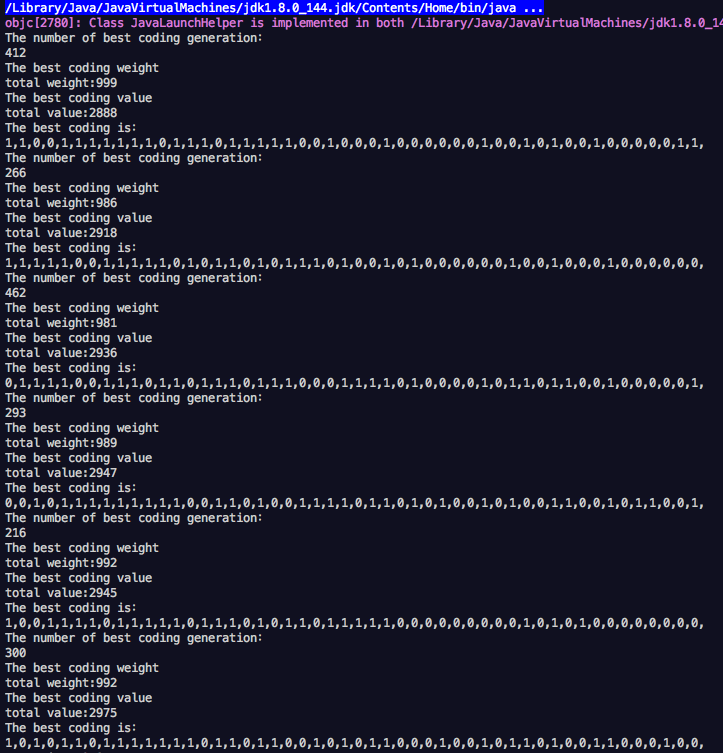
* As for each recursion, we build two new threads to run the code in parallel. The maximum in each two threads in each time recursion will show on the terminal monitor.
* We use default divide number is 10, thus we can get total scale/(divide\*2) populations.

**Testing Process**

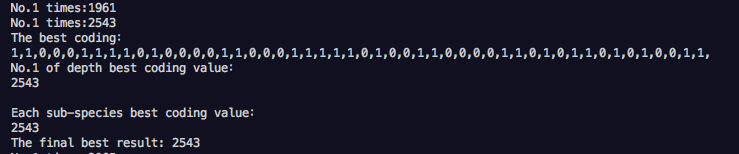
First, we set 50 items and each of them has a value and a weight. Then, to start the test, we set the whole bag capacity is 1000, the maximum of descendants is 500, the rate of crossover is 20%, the rate of mutation for each one is 10% and the rate of mutation for each determined mutation individual is 50%. By adjusting the parameter of the scale of population, we get the following results in one thread running and multiple threads running.



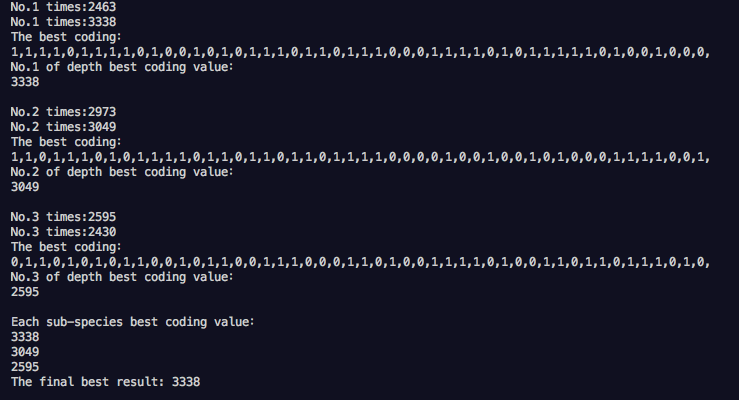
(The result of whole test after running)



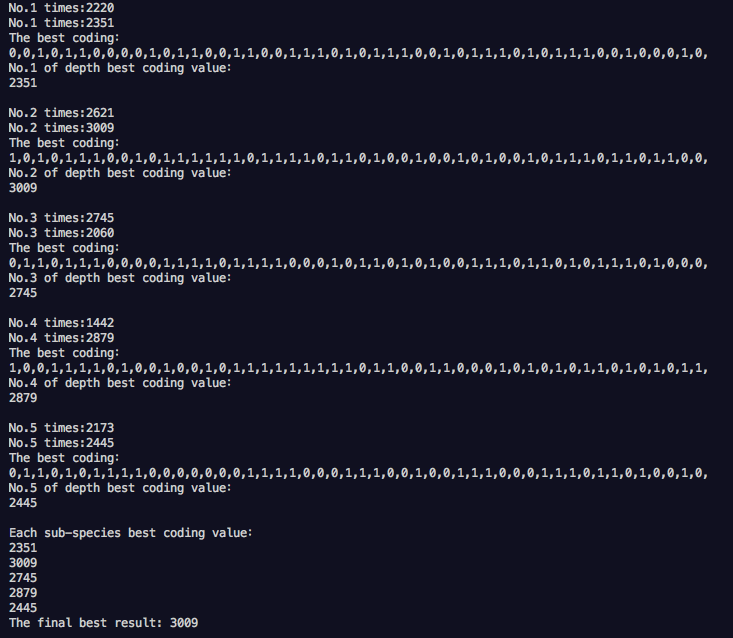
(The results of the algorithm running in one thread)



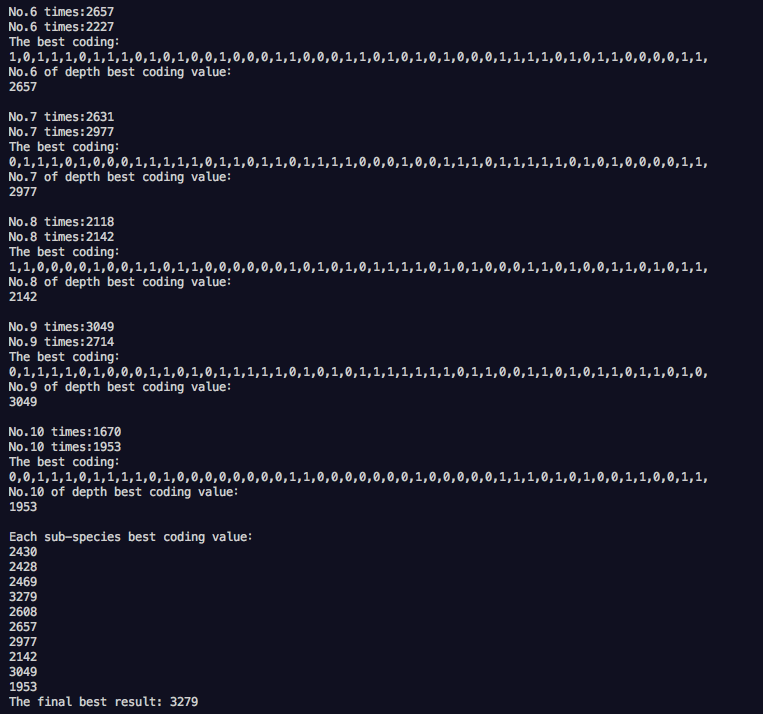
(The result of the algorithm running in multiple threads as scale=20)



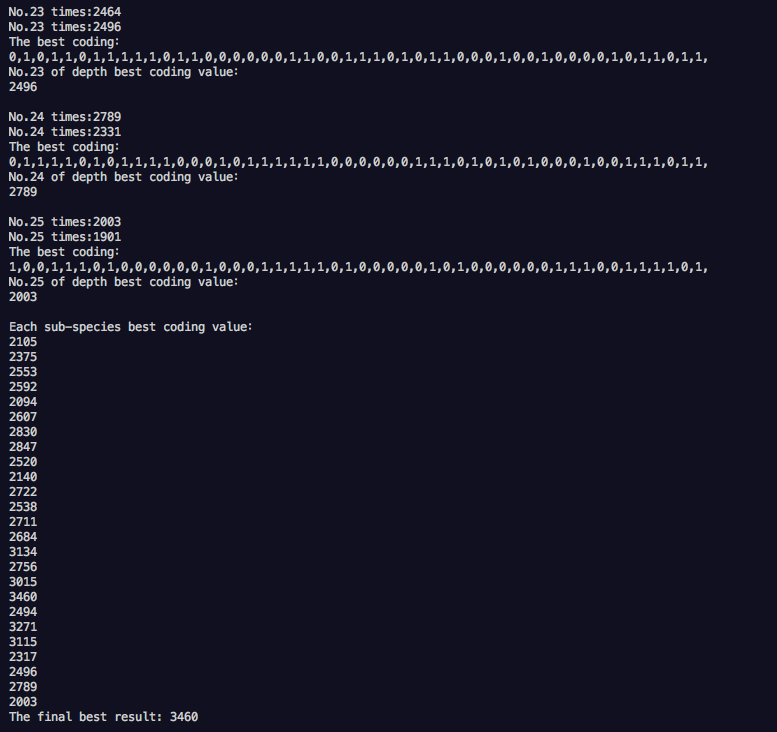
(The result of the algorithm running in multiple threads as scale=60)



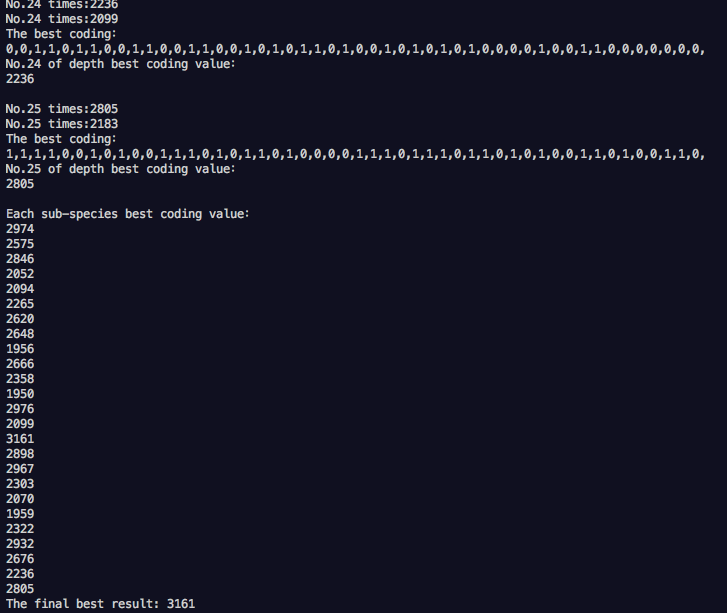
(The result of the algorithm running in multiple threads as scale=100)



(The result of the algorithm running in multiple threads as scale=200)



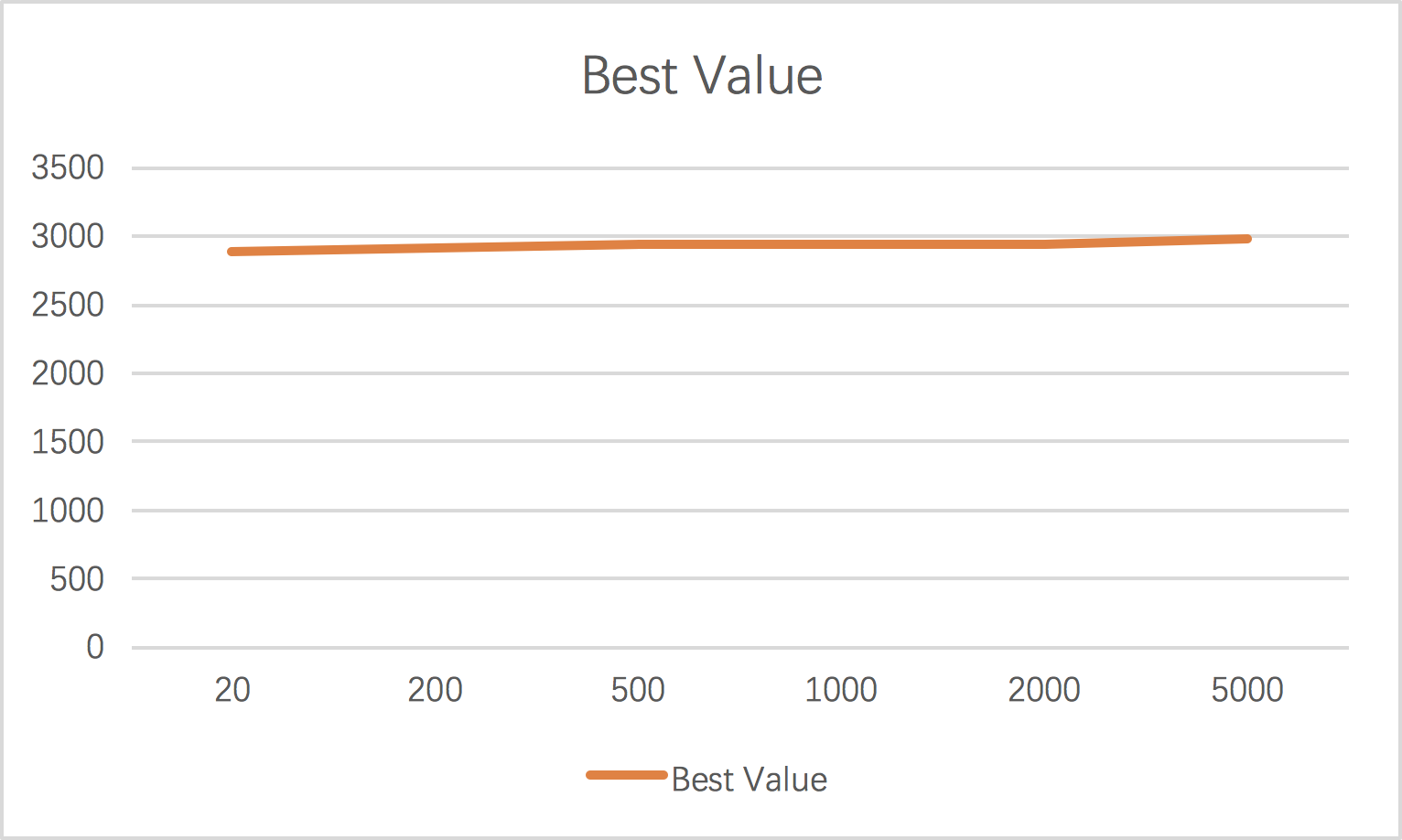
(The result of the algorithm running in multiple threads as scale=500)



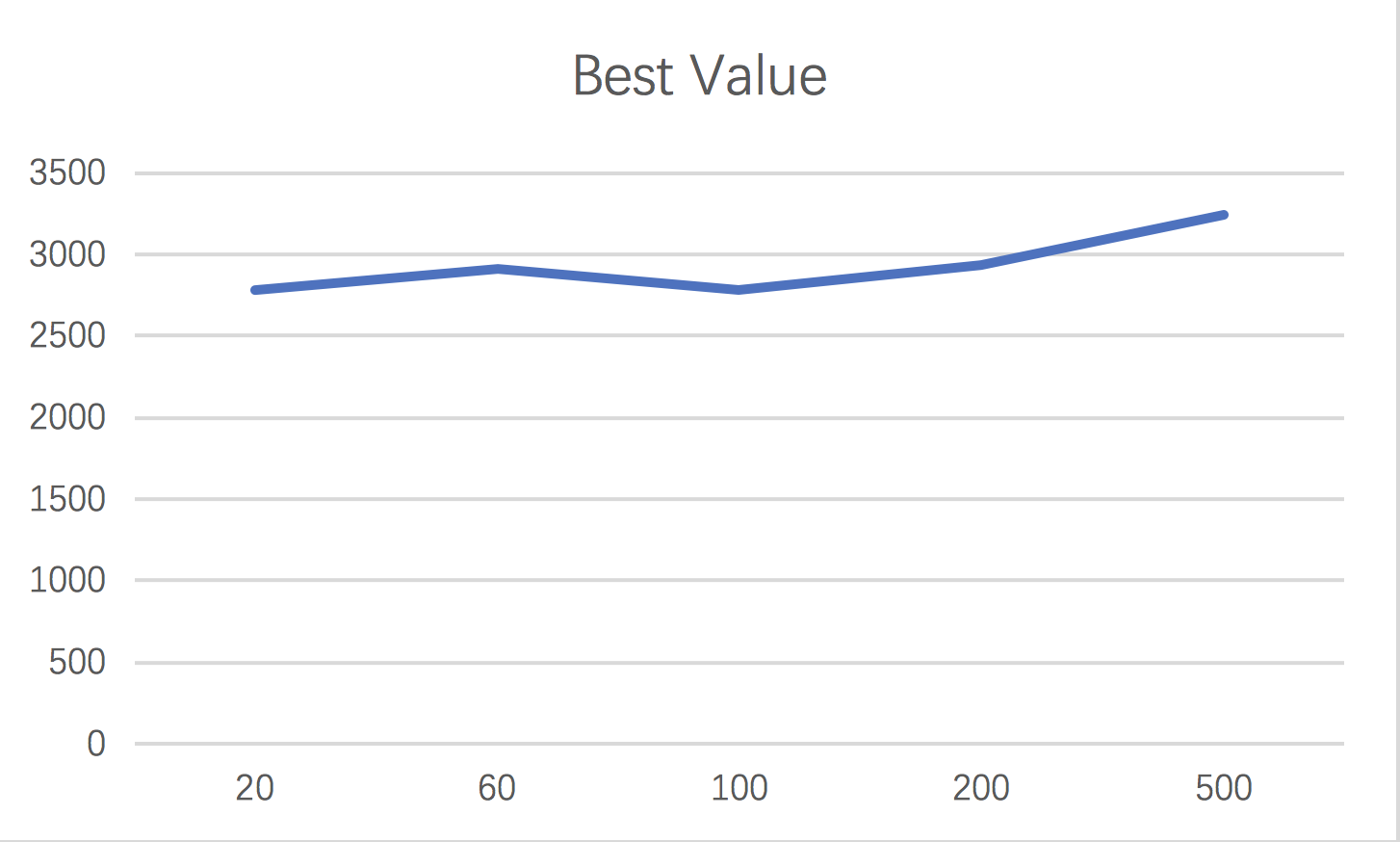
(The result of the algorithm running in multiple threads

as scale=500 and maximum of descendants=1000)

Based on the result data above, we get the following line chart:



(The Best Value results get in one thread running program)



(The Best Value results get in multiple threads running program)

**Conclusion**

Our problem is to find the maximum of total value items that can be put into the limit capacity bag. We have developed the method for solving this problem. The result of running the algorithm in one thread and multiple thread are relatively close and both ways can get a reasonable result. According to our tests, the genetic algorithm will generate a result which is quite close to the optimal result. In one thread running, the program generates the result which is close to the optimal result (or maybe is the optimal result). As for multiple threads running, the program will need more generations to reach a result which is close to the optimal. According the record generated by recBest method, we can see the result can gradually approach the optimal result but can hardly be the actual optimal result. After several generations, the best DNA structure remained unchanged and the crossover between other type influenced little to the result, which is still a problem that needs to be solved.

Genetic algorithms can, in some cases, help produce relatively optimal results with high time complexity. However, since it is difficult to know whether the results are optimal, extra efforts are needed to prove this. We believe that genetic algorithms can easily fall into a situation where the result is only a local optimal solution rather than an overall optimal situation. The population is very likely to be precocious, it is difficult to generate new genetics, and a large amount of space and computation are needed to achieve optimal results. The result is also unstable, and usually we need to run the program multiple times to see if the results are close to optimal.

To sum up, we believe that the genetic algorithm is a good way to get a result close to the optimal solution. With the help of other optimization, the algorithm will help people solve various kinds of problems in our real life.