2016

Knapsack Problem

10.27

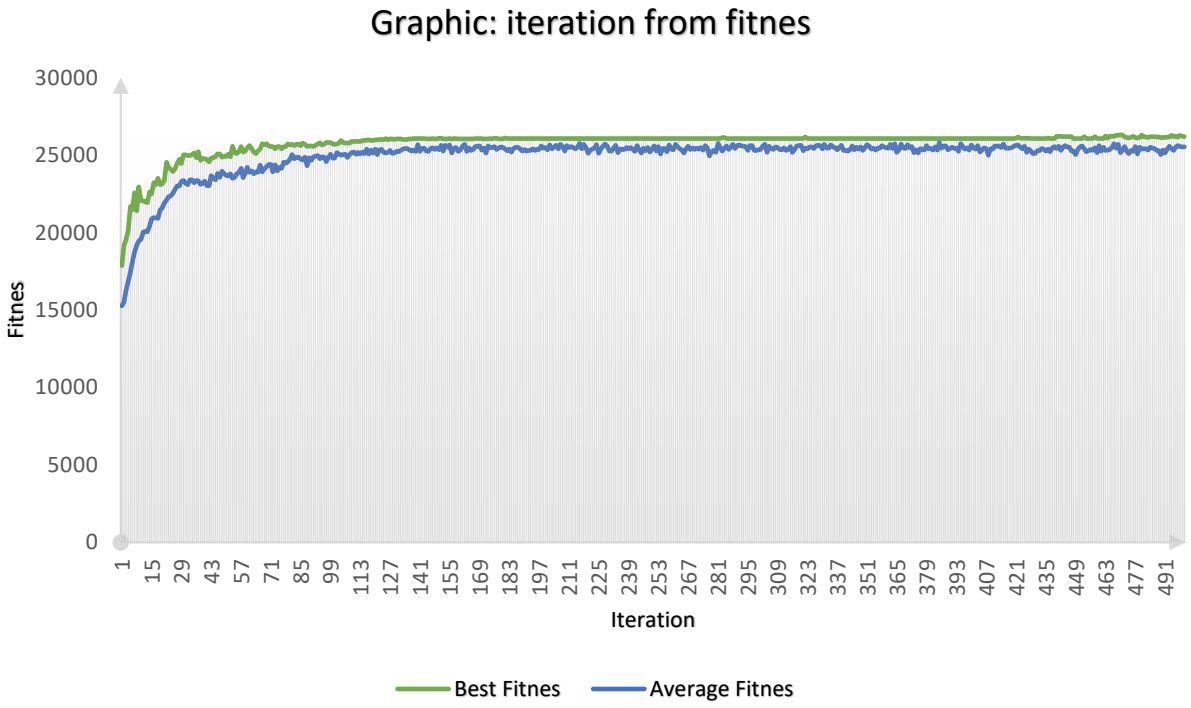
KIRILL TSIBIKOV

My parameters:

A (size of knapsack) = 10000
limit of n = 350

Random generated w_i and x_i:

Volume	1	2	3	3	9	6	10	5	7	2	6	1	1	6	4	1	1	9	6	6
Price	9	4	10	7	7	10	1	7	7	3	1	7	5	6	4	3	8	10	9	9



Tables with 5 best chromosomes:

1 Iteration

```
Best Five chromosome
238  221  78  155  32  211  16  298  51  60  44  331  219  46  230  129  138  65  22  113
275  24  49  250  106  58  77  317  104  201  34  281  6  95  64  35  208  6  239  91
232  11  26  341  7  256  267  47  112  332  48  246  259  23  61  156  284  22  50  63
324  74  202  201  35  29  100  229  190  92  14  151  126  16  23  91  73  106  275  6
347  296  331  141  69  83  53  101  3  35  68  97  145  36  11  64  60  118  33  230
```

100 Iteration

```
Best Five chromosome
342  348  341  312  10  282  19  101  3  199  3  345  345  6  52  320  313  21  125  247
342  345  341  312  10  282  19  101  3  199  3  345  345  6  52  320  313  21  125  247
342  345  341  312  10  282  19  101  3  263  3  316  345  6  52  320  313  21  125  247
342  345  341  312  10  282  19  94  3  310  3  316  345  6  75  320  313  17  107  247
342  345  341  312  10  282  19  94  3  263  3  316  345  6  52  320  313  21  125  247
```

200 Iteration

```
Best Five chromosome
342  345  347  312  10  282  19  101  3  199  3  345  345  13  55  320  346  21  125  247
342  345  347  312  10  282  19  101  3  199  3  345  345  13  55  320  346  21  125  247
342  345  347  312  10  282  19  101  3  199  3  345  345  13  55  320  346  21  125  247
342  345  347  312  10  282  19  101  3  199  3  345  345  13  55  320  346  21  125  247
342  345  347  312  10  282  19  101  3  199  3  345  345  13  55  320  346  21  125  247
```

300 Iteration

```
Best Five chromosome
342  345  347  312  10  282  19  101  3  199  3  346  345  13  55  320  346  21  125  247
342  345  347  312  10  282  19  101  3  199  3  346  345  13  55  320  346  21  125  247
342  345  347  312  10  282  19  101  3  199  3  346  345  13  55  320  346  21  125  247
342  345  347  312  10  282  19  101  3  199  3  346  345  13  55  320  346  21  125  247
342  345  347  312  10  282  19  101  3  199  3  346  345  13  55  320  346  21  125  247
```

500 Iteration

```
Best Five chromosome
342  345  347  312  10  282  2  101  3  198  12  349  345  28  61  317  346  21  125  247
342  345  347  312  10  282  2  101  17  198  3  349  345  13  61  317  346  21  125  247
342  345  347  312  12  282  2  101  3  198  12  349  345  20  61  317  346  21  125  247
342  345  347  312  12  282  2  101  3  198  3  349  345  20  61  317  346  21  125  247
342  345  347  312  12  282  2  101  3  198  3  349  345  20  61  317  346  21  125  247
```

Source Code

```
package brestterespol;

import java.util.Random;

class GeneticAlgo {

    public enum SelectionType {
        TOURNEY, ROULETTE_WHEEL, TRUNCATING
    }

    public enum CrossingType {
        UNIFORM, TWO_POINT_RECOMBINATION, ELEMENTWISE_RECOMBINATION, ONE_ELEMENT_EXCHANGE
    }
    private SelectionType slctp;//Тип Селекции
    private CrossingType crstp;//Тип Скрещивания
    private static int genomLength = 20; //Длина хромосомы
    private static int generationCount = 10000000; //Кол-во поколений
    private static int individualCount = 40; //Кол-во Геномов(Индивидов,Особей) в поколении
    private static int limitVolumes = 10000; //ограничение по общему объему продуктов
    private static int limitPrice = 10; //ограничение по цене продукта
    private static int limitVolume = 10; //ограничение по объему, 1 продукта
    private int limitCount = 350; //лимит по количству
    private int[][] population; //поплуция. где каждый бит в особи = количество данного продукта
    int[] price;
    int[] volume;
    int[] sortFitnes;

    public GeneticAlgo(String s, String c) {
        slctp = SelectionType.valueOf(s);
        crstp = CrossingType.valueOf(c);
        population = new int[individualCount + 1][genomLength];
        volume = new int[genomLength];
```

```
        price = new int[genomLength];
        sortFitnes = new int[individualCount];
    }

    public void run() {
        this.generateFirstGeneration();
        for (int i = 0; i < generationCount; i++) {
            this.selection();
        }
    }

    private void generateFirstGeneration() {
        Random rnd = new Random();
        for (int j = 0; j < genomLength; j++) {
            volume[j] = Math.abs(rnd.nextInt()) % limitVolume + 1;
            price[j] = Math.abs(rnd.nextInt()) % limitPrice + 1;
        }
        for (int i = 0; i < individualCount; i++) {
            do {
                for (int j = 0; j < genomLength; j++) {
                    population[i][j] = Math.abs(rnd.nextInt()) % limitCount;
                }
            } while (!getLimit(i));
        }
    } //генерация первого поколения

    private void selection() {
        int[][] genomListOffsprings = new int[individualCount][genomLength];
        Random rndd = new Random();
        switch (this.slctp) {
            case TRUNCATING: {
                this.sort();
                this.print();
                for (int i = 0; i < GeneticAlgo.individualCount; i++) {
                    do {
                        genomListOffsprings[i] = this.crossing((Math.abs(rndd.nextInt())) %
individualCount / 2, (Math.abs(rndd.nextInt())) % individualCount / 2);
                        population[individualCount] = genomListOffsprings[i];
                    } while (!getLimit(individualCount));
                }
                System.arraycopy(genomListOffsprings, 0, population, 0,
GeneticAlgo.individualCount);
                break;
            }
            default:
                break;
        }
    } //Процедура селекци

    private void sort() {
        for (int i = 0; i < GeneticAlgo.individualCount; i++) {
            sortFitnes[i] = fitnes(i);
        }
        for (int i = 0; i < GeneticAlgo.individualCount; i++) {
            int[] genom;
            for (int j = i; j < GeneticAlgo.individualCount; j++) {
                if (sortFitnes[i] < sortFitnes[j]) {
                    int cur = sortFitnes[i];
                    sortFitnes[i] = sortFitnes[j];
                    sortFitnes[j] = cur;
                    genom = population[i];
                    population[i] = population[j];
                    population[j] = genom;
                }
            }
        }
```

```
    }

    private int fitnes(int nomber) {
        int ftns = 0;
        for (int j = 0; j < genomLength; j++) {
            ftns += price[j] * population[nomber][j];
        }
        return (getLimit(nomber)) ? ftns : 0;
    } //Фитнес функция

    private void print() {
        System.out.print(sortFitnes[0]);
        System.out.print(":");
        System.out.print(avgfitnes());
        System.out.println();
    }

    private int[] crossing(int a, int b) {
        int[] vec = new int[genomLength];
        Random rand = new Random();
        switch (crstp) {
            case UNIFORM: {
                for (int i = 0; i < genomLength; i++) {
                    boolean n = rand.nextBoolean();
                    vec[i] = (n) ? population[b][i] : population[a][i];
                }
                break;
            }
            default:
                break;
        }
        //мутация
        for (int i = 0; i < genomLength; i++) {
            int t = Math.abs(rand.nextInt()) % limitCount;
            vec[i] = ((Math.abs(rand.nextInt()) % 20) == 0) ? (vec[i] == t) ? (t + 1) % limitCount
: t : vec[i];
        }
        return vec;
    } //Процедура скрещивания

    private boolean getLimit(int n) {
        int sum = 0;
        for (int i = 0; i < genomLength; i++) {
            sum += population[n][i] * volume[i];
        }
        return (sum < limitVolumes);
    }

    private int getVolume(int i) {
        int sum = 0;
        for (int j = 0; j < genomLength; j++) {
            sum += population[i][j] * volume[j];
        }
        return sum;
    }

    private int avgfitnes() {
        int sum = 0;
        for (int i = 0; i < individualCount; i++) {
            sum += sortFitnes[i];
        }
        return sum / individualCount;
    }
}

public class BrestTerespol {
```

```
    public static void main(String[] args) {
        GeneticAlgo p = new GeneticAlgo("TRUNCATING", "UNIFORM");
        p.run();
    }
}
```