28.10.2016

Task 5: “Parate Optimal Solutions”.

Student: Maxim Malcev

Group: AI – 10

Algorithm:

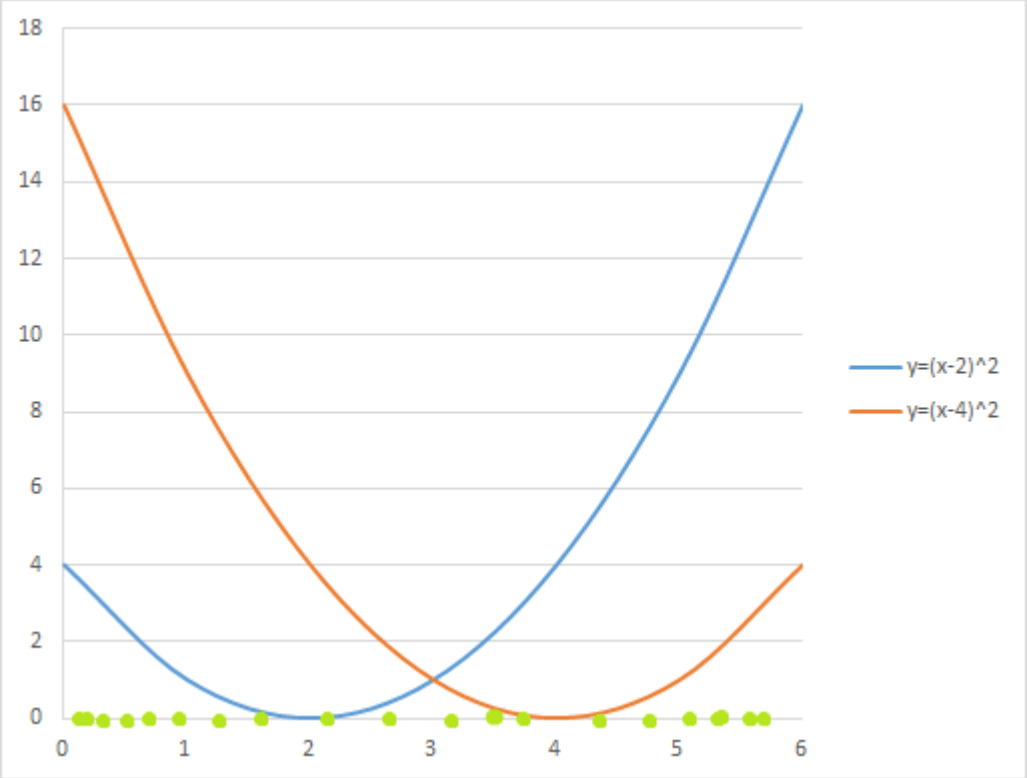
$y_1 = (x-2)^2$ and $y_2 = (x-4)^2$;

1. Create twenty 10-bit binary chromosomes. Each chromosome represent x-cordinate ranges from 0 to 6.
2. Calculate y_1 and y_2 for each of 20 x's represented by these 20 chromosomes.
3. Select individuals uniformly from population.
4. Perform crossover and mutation to create a child.
5. Calculate the rank of the new child.
6. Find the individual in the entire population that is most similar to the child. Replace that individual with the new child if the child's ranking is better, or if the child dominates it.
7. Update the ranking of the population if the child has been inserted.
8. Perform steps 2-6 according to the population size.

Results:

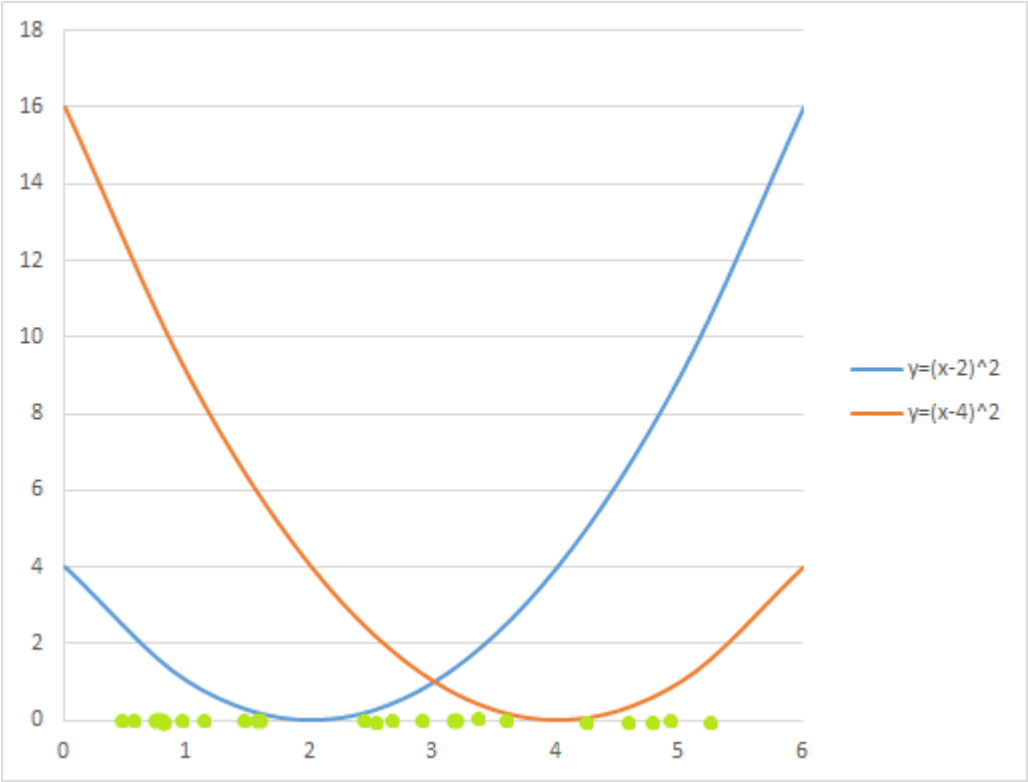
First iteration:

№	Chromosome	x	y_1	y_2	Rank
1	0000011101	0.167784	3.357015	14.68587947	8
2	1001011110	3.556478	2.422624	0.196711764	0
3	0001101100	0.634562	1.864421	11.32617293	1
4	1111001010	5.691473	13.62697	2.86108091	8
5	0000101010	0.244221	3.08276	14.1058759	3
6	1010001000	3.801497	3.245391	0.039403441	2
7	0000111011	0.345692	2.736735	13.35396696	3
8	0010000110	0.783452	1.479989	10.34618104	0
9	1011111110	4.492785	6.213977	0.242837056	8
10	0011100101	1.345764	0.428025	7.044968744	5
11	1101000001	4.884578	8.32079	0.782478238	1
12	1000100101	3.218965	1.485876	0.610015671	6
13	1101110110	5.197356	10.22309	1.433661391	5
14	0100010110	1.628947	0.13768	5.621892329	2
15	1110101000	5.490142	12.18109	2.22052318	5
16	0111001011	2.690142	0.476296	1.71572798	8
17	1110100011	5.460145	11.9726	2.132023421	8
18	0010011000	0.891336	1.229137	9.663794974	4
19	1110110110	5.571457	12.75531	2.469477103	4
20	0101110010	2.170447	0.029052	3.34726418	5



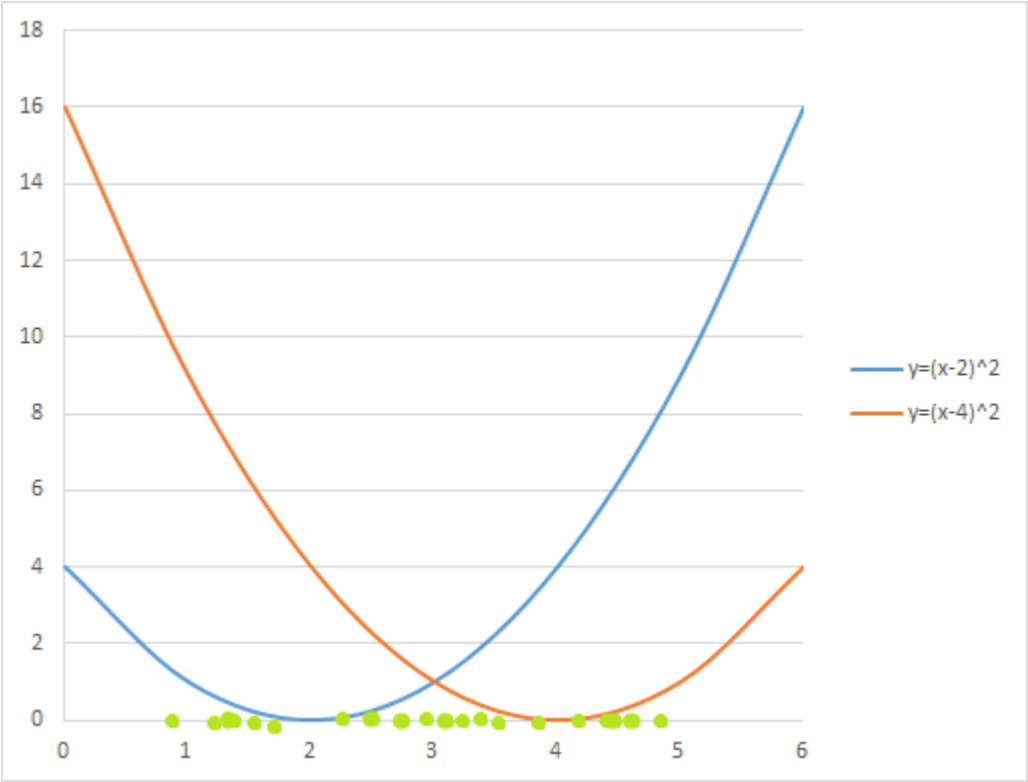
Intermediate iteration 1:

№	Chromosome	x	y_1	y_2	Rank
1	0001010111	0.510244	2.219373	12.17839694	3
2	0001110110	0.690145	1.71572	10.95514012	5
3	0110100000	2.441423	0.194854	2.429162265	5
4	0100000011	1.521454	0.229006	6.143190274	5
5	1000100001	3.194756	1.427442	0.6484179	1
6	0011010001	1.224736	0.601034	7.70209027	0
7	1100000101	4.532475	6.41343	0.283529626	3
8	0111001110	2.712145	0.507151	1.658570501	4
9	1001101000	3.612478	2.600085	0.1501733	4
10	1101000101	4.910147	8.468956	0.828367562	5
11	0001111110	0.741785	1.583105	10.61596499	4
12	1101010010	4.987574	8.925598	0.975302405	4
13	0010101000	0.985473	1.029265	9.087373034	3
14	1101111010	5.220145	10.36933	1.488753821	5
15	0010001011	0.812473	1.41022	10.16032838	4
16	1000101010	3.251425	1.566065	0.560364531	0
17	0110101001	2.493248	0.243294	2.27030159	2
18	0001100101	0.590142	1.9877	11.62713158	0
19	0111111000	2.954723	0.911496	1.092604007	2
20	0100011011	1.662254	0.114072	5.465056361	5



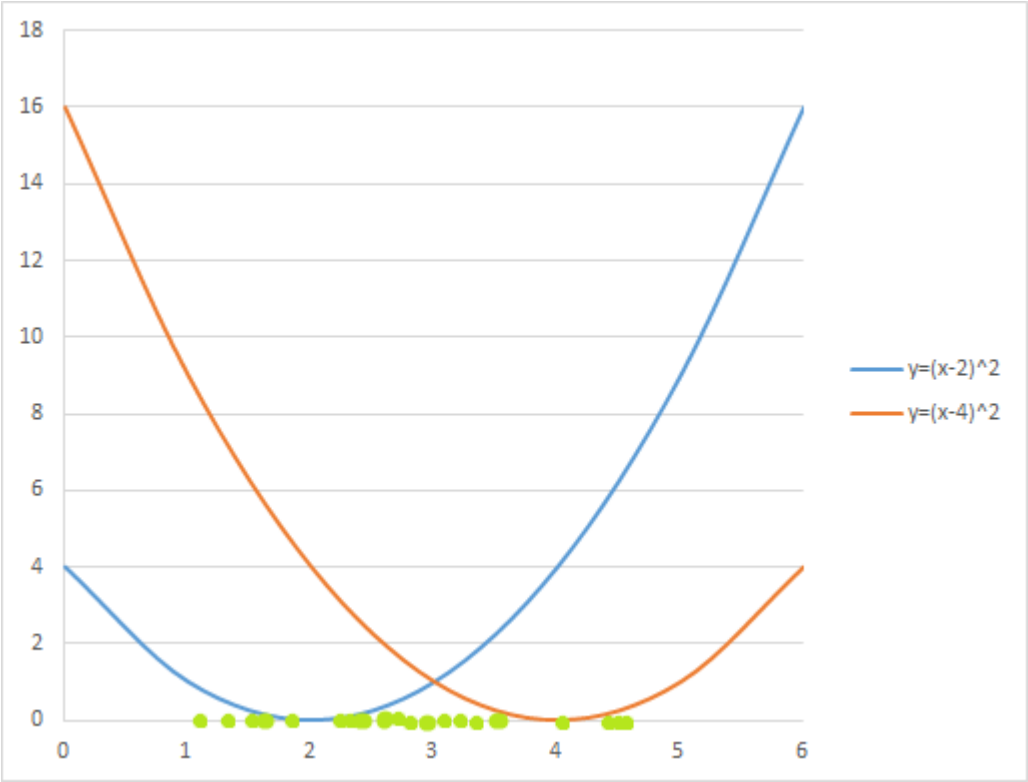
Intermediate iteration 2:

№	Chromosome	x	y_1	y_2	Rank
1	0011010110	1.256475	0.552829	7.526929426	2
2	1001101111	3.652147	2.72959	0.12100171	3
3	1101001100	4.95201	8.714363	0.90632304	5
4	0011110011	1.422542	0.333458	6.643289742	4
5	1100010110	4.632014	6.927498	0.399441696	1
6	0011110111	1.445789	0.30715	6.523993833	0
7	1011111110	4.490111	6.200653	0.240208792	3
8	0010101000	0.985472	1.029267	9.087379063	2
9	0110100010	2.451452	0.203809	2.398000908	4
10	1001001011	3.445721	2.090109	0.30722521	3
11	0111011011	2.784572	0.615553	1.477265223	4
12	0110000000	2.251457	0.063231	3.057402623	4
13	1011111101	4.485754	6.178973	0.235956949	3
14	1001010111	3.514574	2.293934	0.235638401	5
15	1011011001	4.274147	5.171745	0.075156578	1
16	0111111111	2.995424	0.990869	1.00917294	0
17	0100001001	1.554214	0.198725	5.981869158	2
18	0011101001	1.365475	0.402622	6.940721976	0
19	1010100101	3.914756	3.666291	0.00726654	2
20	1000001111	3.089745	1.187544	0.828564165	4



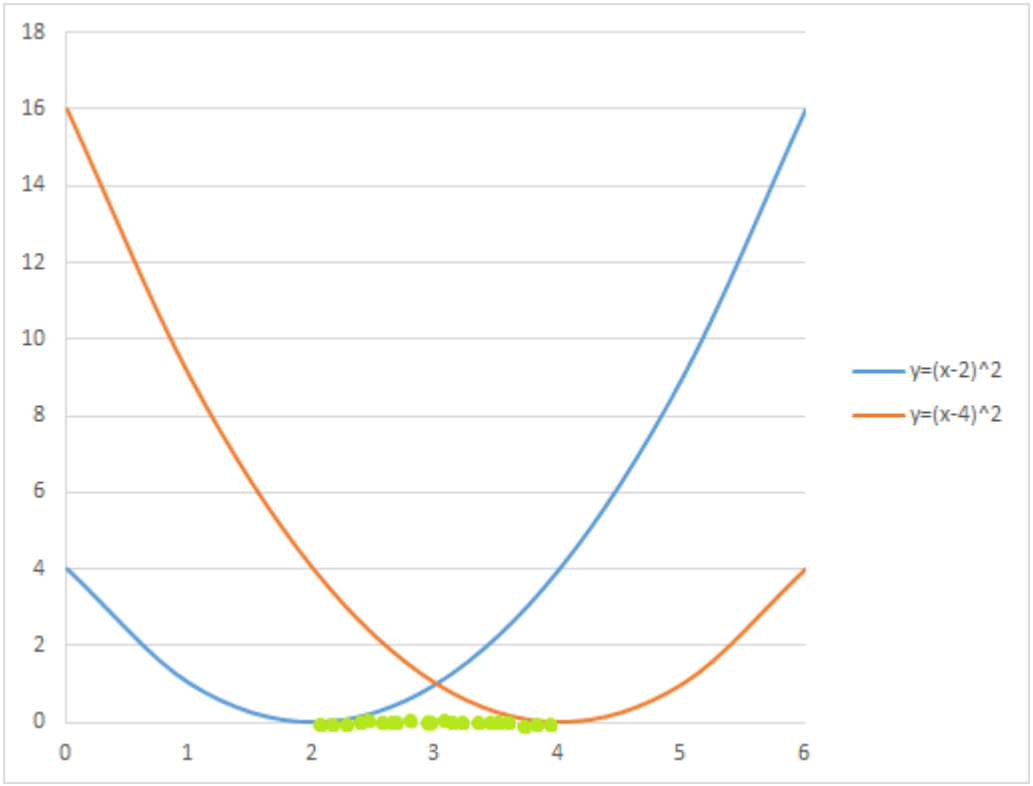
Intermediate iteration 3:

№	Chromosome	x	y_1	y_2	Rank
1	1100001101	4.582554	6.669585	0.339369163	2
2	0100001001	1.554745	0.198252	5.979272015	0
3	0110000010	2.265784	0.070641	3.007505135	2
4	1000100100	3.214793	1.475722	0.616550033	2
5	0101000011	1.896589	0.010694	4.424337835	1
6	0101110111	2.200345	0.040138	3.238758119	0
7	1001111100	3.732547	3.001719	0.071531107	3
8	0101110101	2.185476	0.034401	3.292497347	2
9	0010111100	1.102457	0.805583	8.395755437	4
10	1011111101	4.487485	6.187582	0.237641625	3
11	1100000011	4.521245	6.356676	0.27169635	1
12	1010101110	4.023214	4.093395	0.00053889	0
13	0111011011	2.787845	0.6207	1.469319744	3
14	0011110011	1.424154	0.331599	6.634982616	5
15	1000010011	3.112145	1.236867	0.788286501	1
16	0110100010	2.452141	0.204431	2.395867484	0
17	0111110110	2.941475	0.886375	1.120475176	2
18	0011101000	1.362536	0.40636	6.956216351	0
19	1001110001	3.665242	2.773031	0.112062919	2
20	0111001011	2.689898	0.475959	1.71636725	4

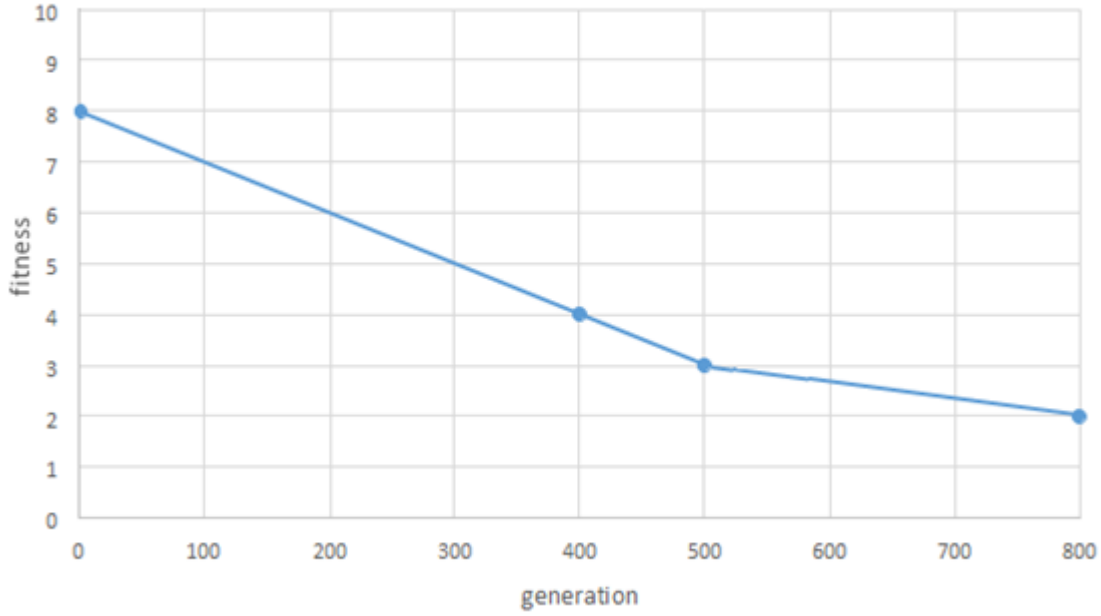


Last iteration:

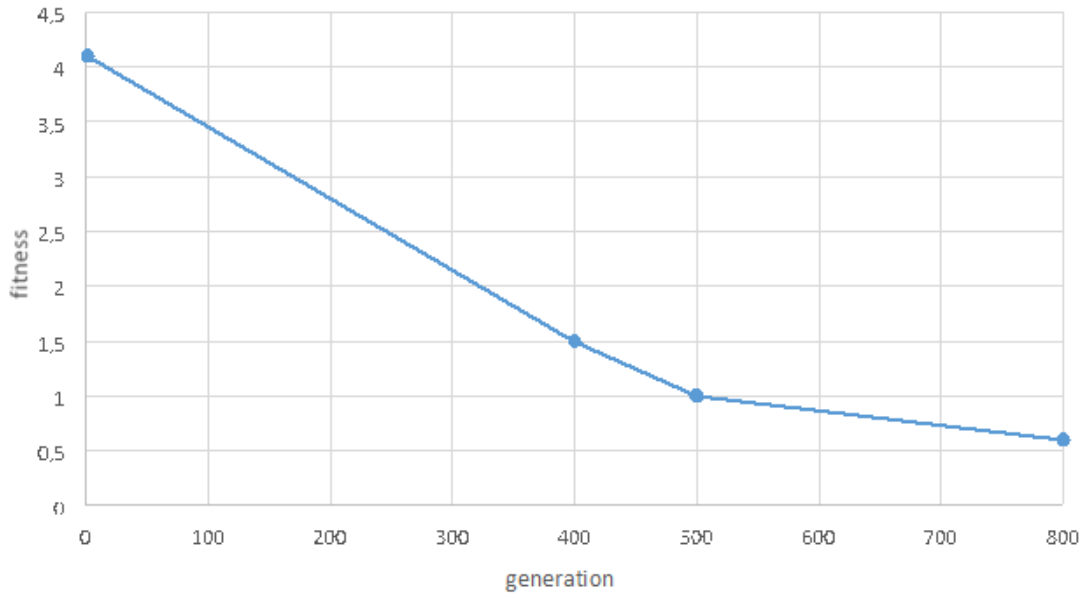
№	Chromosome	x	y_1	y_2	Rank
1	1010100111	3.982521	3.93039	0.000305515	2
2	0011010111	3.001475	1.002952	0.997052176	1
3	1000100100	3.215224	1.476769	0.61587337	0
4	0101101110	2.145457	0.021158	3.439329739	1
5	1010000010	3.765658	3.117548	0.054916173	0
6	0110001010	2.310015	0.096109	2.8560493	1
7	0111010100	2.745862	0.55631	1.572862123	0
8	0110101100	2.510421	0.26053	2.218845597	2
9	1000010101	3.124357	1.264179	0.766750663	0
10	1010011011	3.912457	3.657492	0.007663777	0
11	0101110000	2.156348	0.024445	3.399052697	0
12	1001111110	3.741545	3.032979	0.066798987	0
13	1000000100	3.025121	1.050873	0.950389065	1
14	0110010111	2.385462	0.148581	2.606732953	0
15	0101111011	2.220155	0.048468	3.167848224	1
16	1001010000	3.472458	2.168133	0.278300562	0
17	1010001111	3.842576	3.395086	0.024782316	2
18	0110100010	2.452235	0.204516	2.395576495	0
19	0111111000	2.953225	0.908638	1.095737901	2
20	0101100100	2.085254	0.007268	3.666252245	0



Graph: Maximum fitness vs generation.



Graph: Average fitness vs generation.



Source code:

```
import java.util.Arrays;
import java.util.concurrent.ThreadLocalRandom;

public class Individual {

public static final int GENE_LENGTH = 10;
private int[] genes;
private double x;
private double y1;
private double y2;
private int rank;

public Individual(boolean initialize) {
genes = new int[GENE_LENGTH];
        if (initialize) {
 generateIndividual();
this.x = calculateX();
 this.y1 = calculateY1();
 this.y2 = calculateY2();
this.rank = 0;
        }
    }
    public Individual(int[] genes) {         this.genes = genes;
this.x = calculateX();         this.y1 = calculateY1();         this.y2 =
calculateY2();         this.rank = 0;
    }
    public double getX() {         return x;
    }
    public double getY1() {          return y1;
    }
    public double getY2() {          return y2;
    }      public int getRank() {          return rank;
    }      public int updateRank() {          return ++this.rank;
    }      public void resetRank() {          this.rank = 0;
    }
    public void generateIndividual() {          for (int i = 0; i <
GENE_LENGTH; ++i) {
        genes[i] = ThreadLocalRandom.current().nextInt(0, 2);
    }
```

```
    }
    public double calculateX() {         return
Integer.parseInt(Arrays.toString(genes).replaceAll("[,\\[\\]
]", ""), 2) / 170.5;
    }
    public double calculateY1() {         return Math.pow(x - 2, 2);
    }      public double calculateY2() {         return Math.pow(x - 4, 2);
    }      public int[] getGenesBeforeCutPoint(int cutPoint) {         int[]
genes = new int[cutPoint];
        System.arraycopy(this.genes, 0, genes, 0, cutPoint);         return
genes;
    }
    public int[] getGenesAfterCutPoint(int cutPoint) {         int[] genes =
new int[GENE_LENGTH - cutPoint];
        System.arraycopy(this.genes, cutPoint, genes, 0, genes.length);
return genes;
    }

    @Override
    public boolean equals(Object o) {         if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Individual that = (Individual) o;
        if (Double.compare(that.y1, y1) != 0) return false;         if
(Double.compare(that.y2, y2) != 0) return false;         if
(Double.compare(that.x, x) != 0) return false;         if (rank != that.rank)
return false;         return Arrays.equals(genes, that.genes);

    }

    @Override     public int hashCode() {         int result;         long
temp;
        result = Arrays.hashCode(genes);         temp =
Double.doubleToLongBits(y1);
        result = 31 * result + (int) (temp ^ (temp >>> 32));         temp =
Double.doubleToLongBits(y2);
        result = 31 * result + (int) (temp ^ (temp >>> 32));         temp =
Double.doubleToLongBits(x);
        result = 31 * result + (int) (temp ^ (temp >>> 32));         result =
31 * result + rank;         return result;
    }

    @Override
    public String toString() {         return "Individual{" +
            "genes=" + Arrays.toString(genes) +
            ", x=" + x +
            ", y1=" + y1 +
            ", y2=" + y2 +
            ", rank=" + rank +
            '}';
    } }

    File Population.java:
        import java.util.Arrays;

public class Population {

    public static final int POPULATION_SIZE = 20;
    private Individual[] individuals;

    public Population(boolean initialize) {         individuals = new
Individual[POPULATION_SIZE];

        if (initialize) {
```

```
            for (int i = 0; i < POPULATION_SIZE; ++i) {
individuals[i] = new Individual(true);
            }
        }
    }

    public Population(Individual[] individuals) {         this.individuals =
new Individual[POPULATION_SIZE];         System.arraycopy(individuals, 0,
this.individuals, 0, individuals.length);
    }
    public Individual getIndividual(int index) {         return
individuals[index];
    }
    public void replaceIndividual(Individual individual, Individual child) {
for (int i = 0; i < individuals.length; ++i) {             if
(individuals[i].equals(individual)) {                 individuals[i] = child;
break;
    }
    }
    }
    public Individual[] getAllIndividuals() {         return individuals;
    }
    public void calculateRankForAllIndividuals() {         resetRank();
        for (int i = 0; i < POPULATION_SIZE; ++i) {             for (int j =
0; j < POPULATION_SIZE; ++j) {                 if (i != j &&
(individuals[i].getY1() < individuals[j].getY1() &&
                individuals[i].getY2() < individuals[j].getY2())) {
            individuals[i].updateRank();
        }
        }
        }

    }
    private void resetRank() {
        for (Individual individual: individuals) {
individual.resetRank();
        }
    }

    @Override
    public String toString() {         return "Population{\n" +
Arrays.toString(individuals) + "}\n";
    } }

    File GeneticAlgorithm.java:
        import java.util.concurrent.ThreadLocalRandom;

public class GeneticAlgorithm {
    private Population population;
    public GeneticAlgorithm(Population population) {         this.population
= population;
    }
    public Population run() {
        Population nextGeneration = new
Population(population.getAllIndividuals());
        nextGeneration.calculateRankForAllIndividuals();          for (int i
= 0; i < Population.POPULATION_SIZE; ++i) {
            Individual[] parents = chooseParents(nextGeneration);
int cutPoint = ThreadLocalRandom.current().nextInt(0,
Individual.GENE_LENGTH);
            Individual[] children = crossover(parents, cutPoint);
            Individual child = getBetterIndividualFromTwoChildren(children);
calculateIndividualRank(child);             Individual individual =
findIndividualThatMostSimilarToChild(child);
```

```
            if (childBetterThanParent(individual, child)) {
nextGeneration.replaceIndividual(individual, child);
nextGeneration.calculateRankForAllIndividuals();
            }
        }
        population = nextGeneration;         return population;
    }
    private Individual[] chooseParents(Population fittestIndividuals) {
return new Individual[]

{fittestIndividuals.getIndividual(ThreadLocalRandom.current().nextInt(0,
                    Population.POPULATION_SIZE)),


fittestIndividuals.getIndividual(ThreadLocalRandom.current().nextInt(0,
                           Population.POPULATION_SIZE))};
    }
    private Individual[] crossover(Individual[] parents, int curPoint) {
Individual[] descendants = new Individual[2];

        int[] firstDescendantGenes =
concat(parents[0].getGenesBeforeCutPoint(curPoint),
parents[1].getGenesAfterCutPoint(curPoint));         int[]
secondIndividualGenes = concat(parents[0].getGenesAfterCutPoint(curPoint),
                parents[1].getGenesBeforeCutPoint(curPoint));
        descendants[0] = new Individual(firstDescendantGenes);
descendants[1] = new Individual(secondIndividualGenes);

        return descendants;
    }
    private Individual getBetterIndividualFromTwoChildren(Individual[]
children) {
        return calculateIndividualRank(children[0]) >
                calculateIndividualRank(children[1]) ? children[0] :
children[1];     }
    private Individual findIndividualThatMostSimilarToChild(Individual child)
{
        Individual individual = population.getIndividual(0);         double
minX = Math.abs(population.getIndividual(0).getX() - child.getX());
        for (int i = 1; i < Population.POPULATION_SIZE; ++i) {
double tempMinX = Math.abs(population.getIndividual(i).getX() -
child.getX());
            if (tempMinX < minX) {                 minX = tempMinX;
                individual = population.getIndividual(i);             }
}
        return individual;
    }
    private boolean childBetterThanParent(Individual individual, Individual
child) {
        return child.getRank() > individual.getRank() || (child.getY1() <
individual.getY1() && child.getY2() < individual.getY2());
    }      private int calculateIndividualRank(Individual individual) {
for (int i = 0; i < Population.POPULATION_SIZE; ++i) {             if
(individual.getY1() < population.getIndividual(i).getY1() &&
individual.getY2() < population.getIndividual(i).getY2()) {
individual.updateRank();
    }         }
        return individual.getRank();
    }      private int[] concat(int[] genes1, int[] genes2) {         int[]
genes = new int[Individual.GENE_LENGTH];

        System.arraycopy(genes1, 0, genes, 0, genes1.length);
        System.arraycopy(genes2, 0, genes, genes1.length, genes2.length);
```

```
        return genes;
    }

}
```

File Main.java:

```
        import java.io.FileWriter; import java.io.IOException;


public class Main {
    public static void main(String[] args) throws IOException {
        FileWriter writer = new FileWriter("output.txt");
        Population population = new Population(true);
        GeneticAlgorithm geneticAlgorithm = new GeneticAlgorithm(population);

        for (int i = 0; i < 800; ++i) {             population =
geneticAlgorithm.run();
System.out.println((i + 1));
writer.write( i + 1) + ":\n");
            for (Individual individual: population.getAllIndividuals()) {
                writer.write(individual.toString() + "\n");
                System.out.println(individual);
            }
        }
    }
}
```