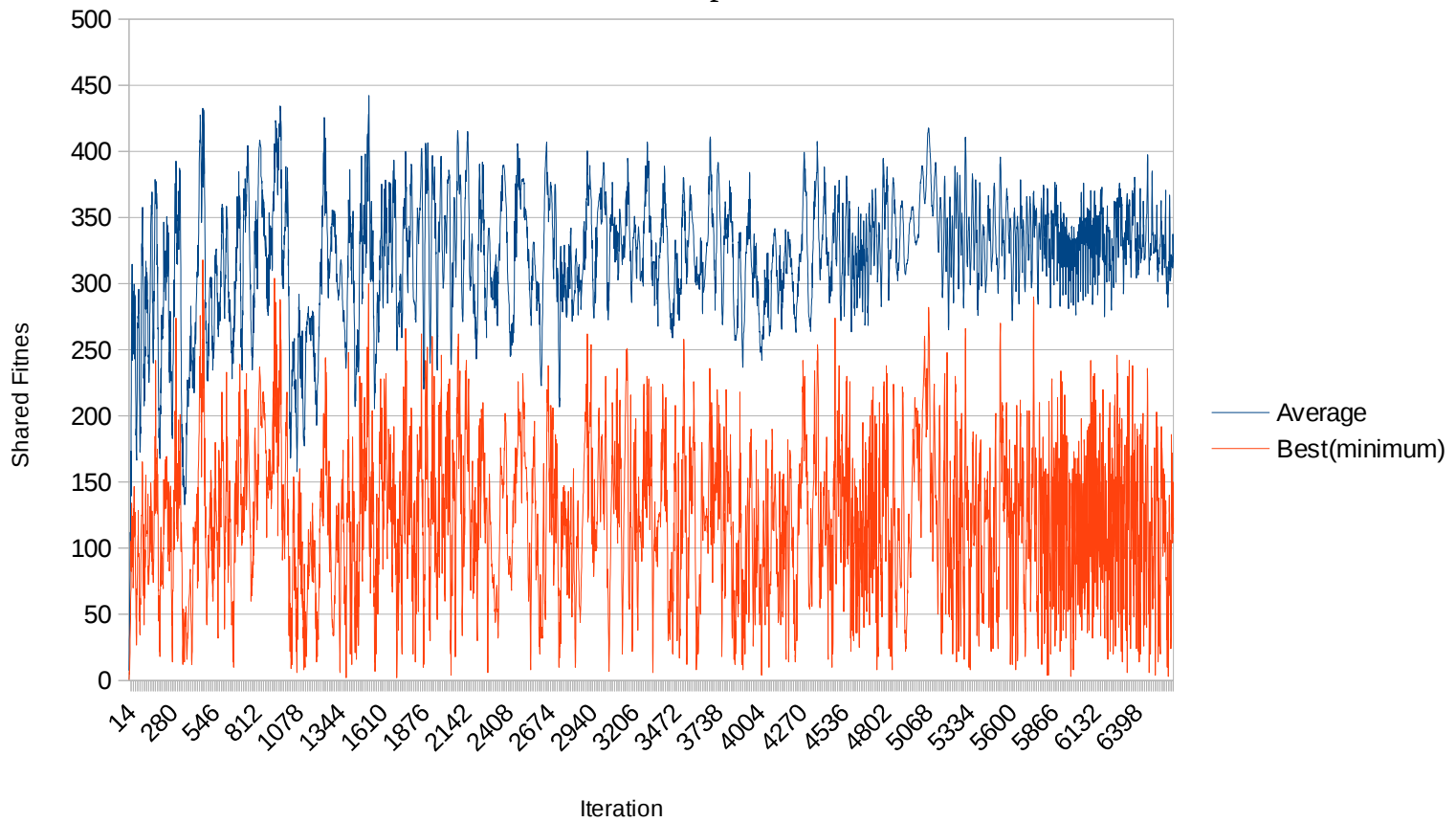


3 - Kirill Tsibikov

Lucky Dog

Brest 2016

Graphic



Tables

0 generation

Position	Original Fitness	Shared
(505;497)	8.0	0.6666666666666666
(499;509)	10.0	0.7692307692307693
(503;509)	12.0	0.9230769230769231
(505;511)	16.0	1.3333333333333333
(485;495)	20.0	1.6666666666666667
(496;516)	20.0	1.8181818181818181
(497;517)	20.0	1.8181818181818181
(493;489)	18.0	2.0
(484;490)	26.0	2.3636363636363638
(474;500)	26.0	2.6
(484;484)	32.0	3.2
(505;523)	28.0	3.5
(487;523)	36.0	4.0
(466;496)	38.0	6.333333333333333
(527;489)	38.0	7.6
(498;540)	42.0	8.4
(521;483)	38.0	12.666666666666666
(459;473)	68.0	17.0
(570;498)	72.0	72.0
(523;433)	90.0	90.0

100 generation

Position	Original Fitness	Shared
(188;22)	410.0	82.0
(178;12)	390.0	97.5
(203;1)	398.0	99.5
(167;21)	388.0	129.33333333333334
(211;25)	414.0	138.0
(246;78)	432.0	144.0
(249;95)	446.0	148.66666666666666
(235;101)	466.0	155.33333333333334
(120;132)	452.0	226.0
(142;120)	462.0	231.0
(89;989)	300.0	300.0
(49;51)	300.0	300.0
(342;122)	380.0	380.0
(257;961)	382.0	382.0
(296;90)	394.0	394.0
(140;72)	412.0	412.0
(49;221)	428.0	428.0
(207;103)	496.0	496.0
(169;177)	546.0	546.0
(202;184)	582.0	582.0

500 generation

Position	Original Fitness	Shared
(540;32)	172.0	86.0
(551;27)	178.0	89.0
(483;159)	276.0	138.0
(473;155)	282.0	141.0
(447;25)	178.0	178.0
(357;119)	362.0	181.0
(355;129)	374.0	187.0
(424;976)	200.0	200.0
(487;89)	202.0	202.0
(510;902)	208.0	208.0
(385;993)	222.0	222.0
(647;7)	254.0	254.0
(433;911)	256.0	256.0
(434;108)	274.0	274.0
(403;921)	276.0	276.0
(531;147)	278.0	278.0
(342;22)	280.0	280.0
(367;63)	296.0	296.0
(436;140)	304.0	304.0
(334;50)	316.0	316.0

Route 5 population best dogs



Source Code

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package squaredog;

import java.awt.*;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*;

/**
 *
 * @author kil
 */
class GeneticAlgo {

    int[][] popln;
    public int[][] crdnt;
```

```

public enum SelectionType {
    TOURNEY, ROULETTE_WHEEL, TRUNCATING
}

public enum CrossingType {
    ONE_POINT_RECOMBINATION, TWO_POINT_RECOMBINATION,
ELEMENTWISE_RECOMBINATION, ONE_ELEMENT_EXCHANGE
}

private SelectionType slctp;
private CrossingType crstp;
private int genomLength; //Длина генома в битах
private int generationCount; //Кол-во поколений
private int individualCount; //Кол-во Геномов (Индивидов, Особей) в поколении
private int[] chosenchromosom;
private SelectionType selectionType; //Тип Селекции
private CrossingType crossingType; //Тип Скрещивания
private double[] ftnes;
private double[] orftn;
private SQUAREDODG d;

public GeneticAlgo(String s, String c, SQUAREDODG dd) {
    slctp = SelectionType.valueOf(s);
    crstp = CrossingType.valueOf(c);
    popln = new int[20][1000];
    crdnt = new int[20][2];
    ftnes = new double[20];
    orftn = new double[20];
    d = dd;
    for (int i = 0; i < 20; i++) {
        crdnt[i][0] = 500;
        crdnt[i][1] = 500;
    }
    for (int i = 0; i < 20; i++) {
        ftnes[i] = 600.D;
    }
    genomLength = 1000;
    generationCount = 20;
    individualCount = 20;
    chosenchromosom = new int[20];
}

public boolean[][] run() {
    this.generateFirstGeneration();
    double ftnsmax = 0.D;
    for (int i = 0; i < 100000000; i++) {
        crdntdog();
        this.selection();
        ftnsmax = ftnes[0];
        for (int j = 1; j < 20; j++) {
            ftnsmax = (ftnsmax >= ftnes[j]) ? ftnes[j] : ftnsmax;
        }
        //      System.out.print(avgftnes());
        //      System.out.print(":");
        //      System.out.print(ftnsmax);
        //      System.out.println();
        if (i == 0 || i == 100 || i == 500 || i == 1000 || i == 5000) {
            System.out.print(i + " generation");
            System.out.println();
            for (int j = 0; j < 20; j++) {
                System.out.print("(" + crdnt[j][0] + ";" + crdnt[j][1] + ")");
            }
        }
    }
}

```

```

        System.out.print(":");
        System.out.print(orftn[j]);
        System.out.print(":");
        System.out.print(ftnes[j]);
        System.out.println();
    }

    }
    d.repaint();
    try {
        Thread.sleep(00);
    } catch (InterruptedException ex) {
        null, ex);
        Logger.getLogger(SQUAREDOD.class.getName()).log(Level.SEVERE,
    }
    if (ftnsmax == 1000) {
        break;
    }
    ftnsmax = 0.D;
}

return (new boolean[20][1000]);
}

private void generateFirstGeneration() {
    Random rnd = new Random();
    for (int i = 0; i < 20; i++) {
        for (int j = 0; j < 1000; j++) {
            popln[i][j] = Math.abs(rnd.nextInt()) % 4 + 1;
        }
    }
} //генерация первого поколения

private void selection() {
    int[][] genomListOffsprings = new int[20][1000];
    Random rndd = new Random();
    switch (this.slctp) {
        case ROULETTE_WHEEL: {
            double[] wheel = new double[this.individualCount];
            wheel[0] = ftnes(0); //Значение ФитнессФункции для 1-ого генома
            this.chosenchromosom[0] = 0;
            for (int i = 1; i < this.individualCount; i++) {
                wheel[i] = wheel[i - 1] + ftnes(i); //Значение
ФитнессФункции для i-ого генома
                this.chosenchromosom[i] = 0;
            }
            double all = wheel[this.individualCount - 1];

            for (int i = 0; i < this.individualCount; i++) {
                double index = Math.abs(rndd.nextFloat()) * all;
                int l = 0;
                int r = individualCount - 1;
                int c = 0;
                while (l < r) {
                    c = (l + r) >> 1;
                    if (index <= wheel[c]) {
                        r = c;
                    } else {
                        l = c + 1;
                    }
                }
            }
        }
    }
}

```

```

    }
    int a = 1;
    index = Math.abs(rndd.nextFloat()) * all;

    l = 0;
    r = individualCount - 1;
    c = 0;
    while (l < r) {
        c = (l + r) >> 1;
        if (index <= wheel[c]) {
            r = c;
        } else {
            l = c + 1;
        }
    }
    this.chosenchromosom[l]++;
    this.chosenchromosom[a]++;
    genomListOffsprings[i] = this.crossing(l, a);
}
popln = genomListOffsprings;
break;
}
case TOURNEY: {
    for (int i = 0; i < this.individualCount; i++) {
        int index1 = rndd.nextInt(individualCount);
        int index2 = rndd.nextInt(individualCount);
        int index3 = rndd.nextInt(individualCount);
        int index4 = rndd.nextInt(individualCount);
        double fr1 = fitnes(index1);
        double fr2 = fitnes(index2);
        index1 = (fr1 > fr2) ? index1 : index2;
        double fr3 = fitnes(index3);
        double fr4 = fitnes(index4);
        index2 = (fr3 > fr4) ? index3 : index4;
        genomListOffsprings[i] = this.crossing(index1, index2);
    }
    popln = genomListOffsprings;
    break;
}
case TRUNCATING: {
    this.sort();
    for (int i = 0; i < this.individualCount; i++) {
        genomListOffsprings[i] =
this.crossing((Math.abs(rndd.nextInt())) % 10, (Math.abs(rndd.nextInt())) % 10);
    }
    popln = genomListOffsprings;
    break;
}
default:
    break;
}
} //Процедура селекци

private int[] crossing(int a, int b) {
    int[] vec = new int[1000];
    switch (crstp) {
        case ONE_ELEMENT_EXCHANGE: {
            for (int i = 0; i < genomLength; i++) {
                Random rndd = new Random();

```

```

        vec[i] = (rndd.nextBoolean()) ? popln[b][i] : popln[a][i];
    }
    break;
}
case ONE_POINT_RECOMBINATION: {
    Random rndd = new Random();
    int point = Math.abs(rndd.nextInt()) % genomLength;
    System.arraycopy(popln[a], 0, vec, 0, point);
    System.arraycopy(popln[b], point, vec, point, genomLength -
point);
    break;
}
default:
    break;
}

//mutation
Random rdd = new Random();
for (int i = 0; i < 1000; i++) {
    vec[i] = (((Math.abs(rdd.nextInt())) % 1000) == 0) ?
(Math.abs(rdd.nextInt()) % 4 + 1 : vec[i];
}
return vec;
} //Процедура скрещивания

private double fitnes(int nmb) {

    double a, b, c, d;
    a = dst(crdnt[nmb][0], crdnt[nmb][1], 200, 200);
    b = dst(crdnt[nmb][0], crdnt[nmb][1], 800, 200);
    c = dst(crdnt[nmb][0], crdnt[nmb][1], 200, 800);
    d = dst(crdnt[nmb][0], crdnt[nmb][1], 800, 800);
    double originalftns = 600 - Math.abs(Math.min(Math.min(a, b),
Math.min(c, d)));
    orftn[nmb]=originalftns;
    double sumdiij = 0.D;
    for (int i = 0; i < 20; i++) {
        sumdiij += ((Math.abs(crdnt[nmb][0] - crdnt[i][0]) <= 25) &&
(Math.abs(crdnt[nmb][1] - crdnt[i][1]) <= 25) && (dst(crdnt[nmb][0], crdnt[nmb]
[1], crdnt[i][0], crdnt[i][1]) < 50)) ? 1 - dst(crdnt[nmb][0], crdnt[nmb][1],
crdnt[i][0], crdnt[i][1]) / 50 : 0;
    }
    return originalftns / sumdiij;
} //Фитнес функция

private int dst(int x1, int y1, int x2, int y2) {
    return Math.abs(x1 - x2) + Math.abs(y1 - y2);
}

private double avgfitnes() {
    double ftns = 0.D;
    for (int i = 0; i < 20; i++) {
        ftns += this.ftnes[i];
    }
    return ftns / 20;
} //Фитнес функция

private void crdntdog() {
    for (int i = 0; i < 20; i++) {
        for (int j = 0; j < 1000; j++) {

```

```

        switch (popln[i][j]) {
            case 1: {
                crdnt[i][1] += (crdnt[i][1] == 1000) ? -999 : 1;
                break;
            }
            case 2: {
                crdnt[i][1] -= (crdnt[i][1] == 0) ? -999 : 1;
                break;
            }
            case 3: {
                crdnt[i][0] += (crdnt[i][0] == 1000) ? -999 : 1;
                break;
            }
            case 4: {
                crdnt[i][0] -= (crdnt[i][0] == 0) ? -999 : 1;
                break;
            }
            default: {
                System.out.print("ERROR!");
                break;
            }
        }
    }
}

private void sort() {
    for (int i = 0; i < this.individualCount; i++) {
        ftnes[i] = ftnes(i);
    }
    for (int i = 0; i < this.individualCount; i++) {
        int[] amiba;
        amiba = new int[1000];
        amiba = popln[i];
        double fit = ftnes[i];
        double ft=orftn[i];
        int[] c = new int[2];
        c = crdnt[i];
        for (int j = i; j < this.individualCount; j++) {
            if (ftnes[j] < fit) {
                fit = ftnes[j];
                ft=orftn[j];
                amiba = popln[j];
                popln[j] = popln[i];
                popln[i] = amiba;
                ftnes[j] = ftnes[i];
                orftn[j]=orftn[i];
                ftnes[i] = fit;
                orftn[i]=ft;
                c = crdnt[j];
                crdnt[j] = crdnt[i];
                crdnt[i] = c;
            }
        }
    }
}

public class SQUARED OG extends JPanel {

```

```

public GeneticAlgo p;

public SQUAREDOG() {

    JFrame frame = new JFrame("SQUAREDOG");
    frame.setMinimumSize(new Dimension(1000, 1000));
    frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    frame.getContentPane().add(this);
    frame.pack();
    Container c = frame.getContentPane();
    this.setLayout(new GridLayout(1000, 1));
    for (int i = 0; i < 1; i++) {
        this.add(new JLabel(" "));
    }
    JScrollPane jsp = new JScrollPane(this);
    c.add(jsp);
    frame.setSize(1000, 1000);
    frame.setVisible(true);
    p = new GeneticAlgo("TRUNCATING", "ONE_ELEMENT_EXCHANGE", this);
    p.run();
    //repaint();
}

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.setColor(Color.orange);
    g.drawRect(0, 0, 1000, 1000);
    g.fillRect(200, 200, 15, 5);
    g.fillRect(200, 800, 15, 5);
    g.fillRect(800, 200, 15, 5);
    g.fillRect(800, 800, 15, 5);
    g.setColor(Color.red);
    for (int i = 0; i < 20; i++) {
        g.fillRect(p.crdnt[i][0], p.crdnt[i][1], 5, 5);
    }
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) throws InterruptedException {
    SQUAREDOG d = new SQUAREDOG();

    // TODO code application logic here
}
}

```