Contemporary Intelligent Information Techniques (CIIT)
Practice #5 (28/10/2016)
Siarhei Savaniuk (AI-10)

LabWork #5 Parate Optimum Solution- Dominate & Rank

Source code (written in Java)

File Individual.java:

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

public class Individual {

    public static final int GENE_LENGTH = 10;

    private int[] genes;
    private double y1;
    private double y2;
    private double x;

    public Individual(boolean initialize, int condition) {
        genes = new int[GENE_LENGTH];

        if (initialize) {
            generateIndividual(condition);
            StringBuilder sb = new StringBuilder();
            for (int gene : genes) {
                sb.append(gene);
            }
            x = Integer.parseInt(sb.toString(), 2) / 170.5;
            y1 = Math.pow(x - 2, 2);
            y2 = Math.pow(x - 4, 2);
        }
    }

    public Individual(int[] genes) {
        this.genes = genes;
        StringBuilder sb = new StringBuilder();
        for (int gene : genes) {
            sb.append(gene);
        }
        x = Integer.parseInt(sb.toString()) / 170.5;
        y1 = Math.pow(x - 2, 2);
        y2 = Math.pow(x - 4, 2);
    }

    public double getY1() {
        return y1;
    }

    public double getY2() {
        return y2;
    }

    public void generateIndividual(int condition) {
        if (condition == 1) {
            for (int i = 0; i < GENE_LENGTH; ++i) {
                genes[i] = 0;
            }
        } else if (condition == 2) {
            for (int i = 0; i < GENE_LENGTH; ++i) {
                genes[i] = 1;
            }
        } else {
```

```
            for (int i = 0; i < GENE_LENGTH; ++i) {
                genes[i] = ThreadLocalRandom.current().nextInt(0, 2);
            }
        }
    }

    public double getX() {
        return x;
    }

    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 String toString() {
        return "Individual{" +
                "y1=" + y1 +
                ", y2=" + y2 +
                ", x=" + x +
                ", genes=" + Arrays.toString(genes) +
                '}' + '\n';
    }
}
```

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) {
            individuals[0] = new Individual(true, 1);
            for (int i = 1; i < POPULATION_SIZE - 1; ++i) {
                individuals[i] = new Individual(true, 0);
            }
            individuals[POPULATION_SIZE - 1] = new Individual(true, 2);
        }
    }

    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 addIndividual(int index, Individual individual) {
        individuals[index] = individual;
    }

    public Individual[] getHalfFittestIndividuals() {
        Individual[] fittestIndividuals = new Individual[POPULATION_SIZE /
2];

        System.arraycopy(individuals, 0, fittestIndividuals, 0,
fittestIndividuals.length);

        return fittestIndividuals;
    }

//    public double getMaxFitness() {
//        return individuals[0].getFitnessValue();
//    }
//
//    public double getAverageFitness() {
//        double sum = 0;
//
//        for (int i = 0; i < POPULATION SIZE; ++i) {
//            sum += individuals[i].getFitnessValue();
//        }
//
//        return sum / POPULATION SIZE;
//    }

    public Individual[] getAllIndividuals() {
        return individuals;
    }

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

File GeneticAlgorithm.java:

```
import java.util.Arrays;
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());

        for (int i = 0, j = Population.POPULATION_SIZE / 2; i <
Population.POPULATION_SIZE / 4; ++i, j += 2) {
            Individual[] parents = chooseParents(nextGeneration);

            int cutPoint = ThreadLocalRandom.current().nextInt(0,
Individual.GENE_LENGTH);

            Individual[] descendants = crossover(parents, cutPoint);

            nextGeneration.addIndividual(j, descendants[0]);
            nextGeneration.addIndividual(j + 1, descendants[1]);
        }
```

```
        population = nextGeneration;

        Arrays.sort(population.getAllIndividuals());

        return population;
    }

    private Individual[] chooseParents(Population fittestIndividuals) {
        return new Individual[]
{fittestIndividuals.getIndividual(ThreadLocalRandom.current().nextInt(0,
20)),

fittestIndividuals.getIndividual(ThreadLocalRandom.current().nextInt(0,
20))};
    }

    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 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:

```
/**
 * Created by SergeiPC on 28.10.2016.
 *
 * 10 ген 20 хромосом; x=0:6  (x-2)^2 (x-4)^2
 */
public class Main {
    public static void main(String[] args) {
        Population population = new Population(true);
        for (Individual individual: population.getAllIndividuals()) {
            System.out.println(individual);
        }
    }
}
```