Contemporary Intelligent Information Techniques (CIIT)
Practice #4 (24/10/2016)
Siarhei Savaniuk (AI-10)

LabWork #4 Iterated Prisoner's Dilemma

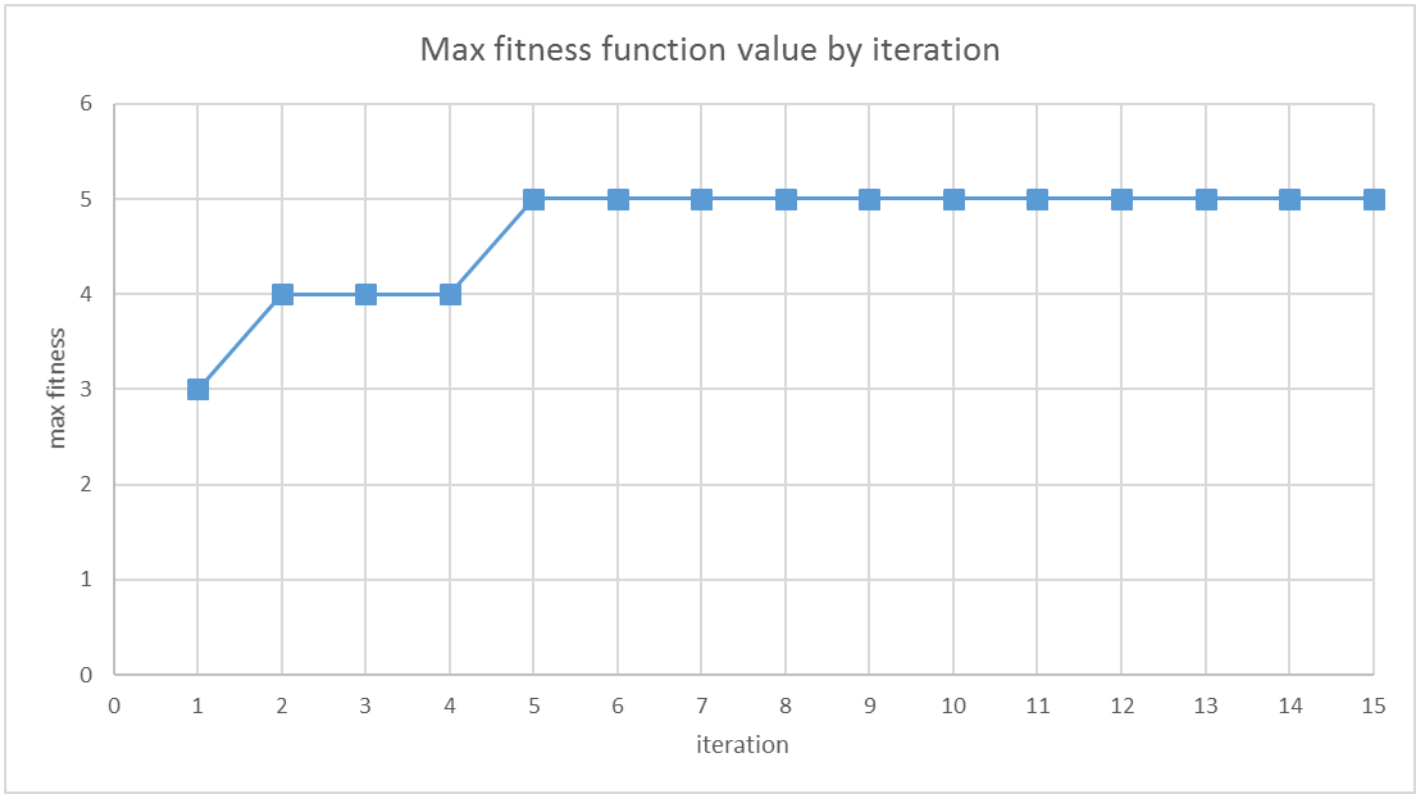

Diagram1 – Max fitness function value by iteration

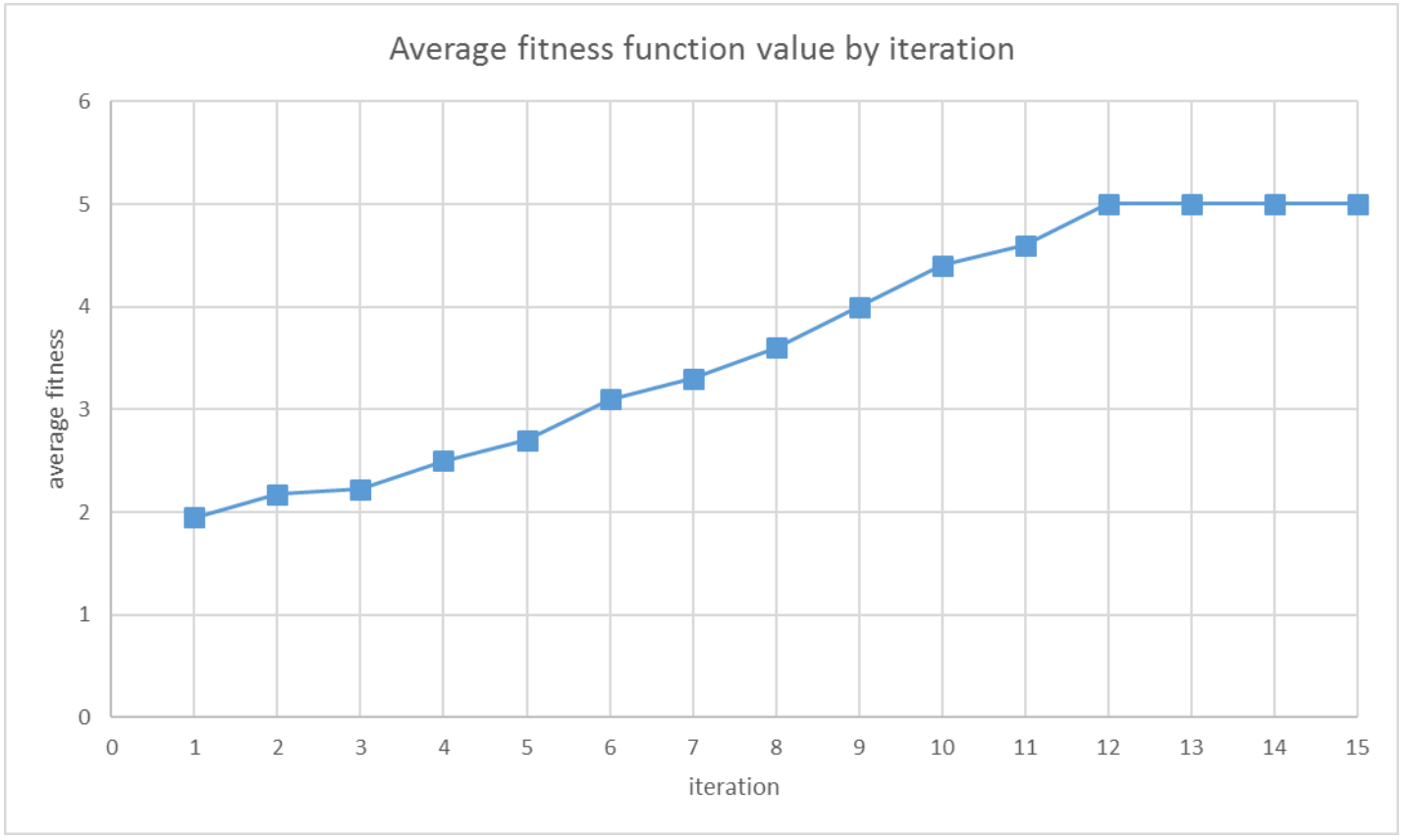

Diagram2 – Average fitness function value by iteration

1. First iteration

Best: 10101001110100010111101010110011100111100001001010101001001000 01

Random: 10001001100100011110100001001100111000010100101001010000011011 01

Game									Result
0	1	0	1	1	0	0	1	1	20
0	1	1	0	0	0	1	1	0	15

Score best: 0 + 1 + 0 + 5 + 5 + 3 + 0 + 1 + 5 = 20

Score random: 0 + 1 + 5 + 0 + 3 + 0 + 5 + 1 + 0 = 15

2. Average iteration:

Best: 10101111000010010101010101011110110001001101010101011011001100 01

Random: 1010110100 0111100001000011010000101000111000001100100111010111 00

Game									Result
1	0	1	0	1	1	1	0	1	25
0	1	0	0	0	1	0	1	1	15

Score best: 5 + 0 + 5 + 3 + 5 + 1 + 5 + 0 + 1 = 25

Score random: 0 + 5 + 0 + 3 + 0 + 1 + 0 + 5 + 1 = 15

3. Last iteration:

Best: 1010 1011111101 0100111010 1111101001 0110111100 0111010011 1100110001

Random: 1111 0011000100 0010111000 1011101010 0110110000 0010010001 1000001100

Game									Result
0	1	1	1	0	1	1	1	1	27
1	0	0	0	1	0	1	0	1	12

Score best: 0 + 5 + 5 + 5 + 0 + 5 + 1 + 5 + 1 = 27

Score random: 5 + 0 + 0 + 0 + 5 + 0 + 1 + 0 + 1 = 12

Code (written in Java):

File Individual.java:

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

public class Individual {

    public static final int GENE_LENGTH = 64;

    private int[] genes;
    private List<List<Integer>> games = new ArrayList<>();
    private int fitnessValue;

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

        if (initialize) {
            generateIndividual();
            fitnessValue = 0;
        }
    }

    public Individual(int[] genes) {
        this.genes = genes;
        fitnessValue = 0;
    }

    public int getFitnessValue() {
        return fitnessValue;
    }
```

```
    public void setFitnessValue(int fitnessValue) {
        this.fitnessValue = fitnessValue;
    }

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

    public int getGene(int index) {
        return genes[index];
    }

    public void addGames(List<Integer> game) {
        games.add(game);
    }

    public void updateFitness(int value) {
        fitnessValue += value;
    }

    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;
    }

    public void resetGames() {
        games.clear();
    }

    public void resetFitnessValue() {
        fitnessValue = 0;
    }

    public List<List<Integer>> getBestMatch() {
        int currentIndividualPoints = 0, maxPoints = 0, index = 0;

        for (int i = 0; i < games.size(); i += 2) {
            currentIndividualPoints = 0;
            for (int j = 0; j < games.get(i).size(); ++j) {
                if (games.get(i).get(j) == 1 && games.get(i + 1).get(j) == 1)
{
                    currentIndividualPoints += 1;
                } else if (games.get(i).get(j) == 1 && games.get(i +
1).get(j) == 0) {
                    currentIndividualPoints += 5;
                } else if (games.get(i).get(j) == 0 && games.get(i +
1).get(j) == 0) {
                    currentIndividualPoints += 3;
                }
            }

            if (currentIndividualPoints > maxPoints) {
                maxPoints = currentIndividualPoints;
                index = i;
            }
```

```
        }
        return games.subList(index, index + 2);
    }

    @Override
    public String toString() {
        return "Individual{" +
                "fitnessValue=" + fitnessValue +
                ", genes=" + Arrays.toString(genes) +
                '}' + '\n';
    }
}
```

File Population.java:

```
import java.util.Arrays;

public class Population {

    public static final int POPULATION_SIZE = 40;

    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 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;
    }

    public void sort() {
        for (int i = 0; i < individuals.length - 1; ++i) {
            for (int j = i + 1; j < individuals.length; ++j) {
                if (individuals[i].getFitnessValue() <
individuals[j].getFitnessValue()) {
                    Individual individual = individuals[i];
                    individuals[i] = individuals[j];
                    individuals[j] = individual;
                }
            }
        }
    }

    public void resetGames() {
        for (Individual individual: individuals) {
            individual.resetGames();
        }
    }

    public void resetFitnessValue() {
        for (Individual individual : individuals) {
            individual.resetFitnessValue();
        }
    }

    @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 halfPopulation = new
Population(population.getHalfFittestIndividuals());
        Population nextGeneration = new
Population(halfPopulation.getAllIndividuals());

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

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

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

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

```
        population = nextGeneration;

        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 PrisonerDilemma.java:

```
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;

public class PrisonerDilemma {

    public void play(Population population) {
        List<Integer> gameOfCurIndividual = new ArrayList<>(),
                      gameOfRandomIndividual = new ArrayList<>();

        for (int i = 0; i < 40; ++i) {
            Individual currentIndividual = population.getIndividual(i);
            preparedData(gameOfCurIndividual, gameOfRandomIndividual);
            for (int j = 0; j < 5; ++j) {
                Individual randomIndividual =
population.getIndividual(ThreadLocalRandom.current().nextInt(0, 40));
                for (int k = 0; k < 6; ++k) {
                    calculateNextResult(currentIndividual, randomIndividual,
gameOfCurIndividual, gameOfRandomIndividual);
                }

                if (isWinningGame(gameOfCurIndividual,
gameOfRandomIndividual)) {
```

```
                    currentIndividual.addGames(new
ArrayList<>(gameOfCurIndividual));
                    currentIndividual.addGames(new
ArrayList<>(gameOfRandomIndividual));
                    currentIndividual.updateFitness(1);
                }

                preparedData(gameOfCurIndividual, gameOfRandomIndividual);
            }
        }
        population.sort();
    }

    public void preparedData(List<Integer> gameOfCurIndividual, List<Integer>
gameOfRandomIndividual) {
        gameOfCurIndividual.clear();
        gameOfRandomIndividual.clear();

        for (int i = 0; i < 3; ++i) {
            gameOfCurIndividual.add(ThreadLocalRandom.current().nextInt(0,
2));
            gameOfRandomIndividual.add(ThreadLocalRandom.current().nextInt(0,
2));
        }
    }

    public void calculateNextResult(Individual currentIndividual, Individual
randomIndividual,
                                    List<Integer> gameOfCurIndividual,
List<Integer> gameOfRandomIndividual) {
        StringBuilder sb1 = new StringBuilder();
        StringBuilder sb2 = new StringBuilder();

        for (int i = gameOfCurIndividual.size() - 1, j = 0; j < 3; ++j, --i)
{
            sb1.append(gameOfRandomIndividual.get(i));
            sb1.append(gameOfCurIndividual.get(i));
        }


gameOfCurIndividual.add(currentIndividual.getGene(Integer.parseInt(sb1.toStri
ng(), 2)));

        for (int i = gameOfCurIndividual.size() - 1, j = 0; j < 3; ++j, --i)
{
            sb2.append(gameOfCurIndividual.get(i));
            sb2.append(gameOfRandomIndividual.get(i - 1));
        }


gameOfRandomIndividual.add(randomIndividual.getGene(Integer.parseInt(sb2.toSt
ring(), 2)));
    }

    public boolean isWinningGame(List<Integer> gameOfCurIndividual,
List<Integer> gameOfRandomIndividual) {
        int currentIndividualPoints = 0, randomIndividualPoints = 0;

        for (int i = 0; i < gameOfCurIndividual.size(); ++i) {
            if (gameOfCurIndividual.get(i) == 1 &&
gameOfRandomIndividual.get(i) == 1) {
                currentIndividualPoints += 1;
                randomIndividualPoints += 1;
            } else if (gameOfCurIndividual.get(i) == 1 &&
gameOfRandomIndividual.get(i) == 0) {
                currentIndividualPoints += 5;
            } else if (gameOfCurIndividual.get(i) == 0 &&
gameOfRandomIndividual.get(i) == 1) {
```

```
                randomIndividualPoints += 5;
            } else {
                currentIndividualPoints += 3;
                randomIndividualPoints += 3;
            }
        }

        return currentIndividualPoints > randomIndividualPoints;
    }
}
```

File Main.java:

```
import java.util.List;

public class Main {

    public static void main(String[] args) {
        PrisonerDilemma prisonerDilemma = new PrisonerDilemma();
        Population population = new Population(true);
        GeneticAlgorithm geneticAlgorithm = new GeneticAlgorithm(population);

        for (int i = 0; i < 50; ++i) {
            System.out.println("ITERATION #" + (i + 1));
            prisonerDilemma.play(population);
            List<List<Integer>> list =
population.getIndividual(0).getBestMatch();
            for (List<Integer> match: list) {
                System.out.println(match);
            }
            System.out.println(population);

            population = geneticAlgorithm.run();

            population.resetGames();
            population.resetFitnessValue();
        }
    }
}
```