

Отже, краща продуктивність вийшла для 10 експерименту.

1. Розробити програмну реалізацію еволюційного методу за умови застосування його для обраної моделі

/\*

\* 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 LibTest;

import PetriObj.ArcIn;

import PetriObj.ArcOut;

import PetriObj.ExceptionInvalidNetStructure;

import PetriObj.ExceptionInvalidTimeDelay;

import PetriObj.PetriNet;

import PetriObj.PetriObjModel;

import PetriObj.PetriP;

import PetriObj.PetriSim;

import PetriObj.PetriT;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

/\*\*

// \*

\* @author julia

\*/

public class Robots {

public static void main(String[] args) throws ExceptionInvalidTimeDelay, ExceptionInvalidNetStructure {

int timeModelling = 150000;

List<Integer> verstat1 = Arrays.asList(5,1,4,3,1,2);

List<Integer> verstat2 = Arrays.asList(16,4,1,5,8,6);

List<Integer> robot1 = Arrays.asList(5,3,1,10,8,4);

List<Integer> robot2 = Arrays.asList(8,21,28,3,10,5);

List<Integer> robot3 = Arrays.asList(1,6,1,14,8,28);

double epsilon = 0.001;

double currentValue = Double.MAX\_VALUE;

double prevValue = 0;

int n = 6;

int iter = 0;

while(Math.abs(currentValue - prevValue) > epsilon)

{

iter++;

System.out.println("Iteration " + iter);

//Пошук показниців життєдіяльності та визначення найліпшого значення

double[] result = findBest(timeModelling,n, verstat1,verstat2,robot1,robot2,robot3);

System.out.println("Productivity: " + result[(int)result[0] + 1]);

if(iter==1)

{prevValue = 0;}

else

{prevValue = currentValue;}

currentValue = result[(int)result[0] + 1];

System.out.println("Verstat1: " + verstat1.get((int)result[0]));

System.out.println("Verstat2: " + verstat2.get((int)result[0]));

System.out.println("Robot1: " + robot1.get((int)result[0]));

System.out.println("Robot2: " + robot2.get((int)result[0]));

System.out.println("Robot3: " + robot3.get((int)result[0]));

//відкидання половини найгірших

ArrayList<Integer> indexesToRemove = new ArrayList<Integer>();

for(int i = 0;i<(n/2.0);i++)

{

int indexToRemove = 0;

double valueToRemove = result[1];

for(int j = 0;j<n;j++)

{

if(result[j+1]<valueToRemove)

{

valueToRemove = result[j+1];

indexToRemove = j;

}

}

indexesToRemove.add(indexToRemove);

result[indexToRemove+1] = Integer.MAX\_VALUE;

}

System.out.println(prevValue);

System.out.println(currentValue);

//кросинговер

List<Integer> verstat11 = new ArrayList<Integer>();

List<Integer> verstat22 = new ArrayList<Integer>();

List<Integer> robot11 = new ArrayList<Integer>();

List<Integer> robot22 = new ArrayList<Integer>();

List<Integer> robot33 = new ArrayList<Integer>();

for(int i=0;i<n;i++)

{

Boolean flag = true;

for(int j=0;j<indexesToRemove.size();j++)

{

if(i==indexesToRemove.get(j))

{

flag = false;

}

}

if(!flag)

{

verstat11.add(verstat1.get(i));

verstat22.add(verstat2.get(i));

robot11.add(robot1.get(i));

robot22.add(robot2.get(i));

robot33.add(robot3.get(i));

}

}

for(int i = 0;i<n/2;i++)

{

if(i==((n/2)-1))

{

verstat11.add(verstat1.get(i));

verstat22.add(verstat2.get(i));

robot11.add(robot1.get(0));

robot22.add(robot2.get(0));

robot33.add(robot3.get(0));

}

else

{

verstat11.add(verstat1.get(i));

verstat22.add(verstat2.get(i));

robot11.add(robot1.get(i+1));

robot22.add(robot2.get(i+1));

robot33.add(robot3.get(i+1));

}

}

//мутація

for(int i=0;i<n;i++)

{

int a = 0; // Начальное значение диапазона - "от"

int b = 100; // Конечное значение диапазона - "до"

int probability = a + (int) (Math.random() \* b); // Генерация 1-го числа

if(probability>=50)

{

int c = 0; // Начальное значение диапазона - "от"

int k = 4; // Конечное значение диапазона - "до"

int prob = c + (int) (Math.random() \* k); // Генерация 1-го числа

int q = 0; // Начальное значение диапазона - "от"

int w = 1; // Конечное значение диапазона - "до"

int mutationType = q + (int) (Math.random() \* w); // Генерация 1-го числа

int mutationValue = 2;

if(mutationType==1)

{

mutationValue \*=-1;

}

switch(prob){

case 0:

if(verstat11.get(i)>1)

{verstat11.set(i, verstat11.get(i)+mutationValue);}

break;

case 1:

if(verstat22.get(i)>1)

{verstat22.set(i, verstat22.get(i)+mutationValue);}

break;

case 2:

if(robot11.get(i)>1)

{robot11.set(i, robot11.get(i)+mutationValue);}

break;

case 3:

if(robot22.get(i)>1)

{robot22.set(i, robot22.get(i)+mutationValue);}

break;

case 4:

if(robot33.get(i)>1)

{robot33.set(i, robot33.get(i)+mutationValue);}

break;

}

}

}

//створити нове покоління

List<Integer> verstat111 = new ArrayList<Integer>();

List<Integer> verstat222 = new ArrayList<Integer>();

List<Integer> robot111 = new ArrayList<Integer>();

List<Integer> robot222 = new ArrayList<Integer>();

List<Integer> robot333 = new ArrayList<Integer>();

for(int i=0;i<n;i++)

{

verstat111.add(verstat11.get(i));

verstat222.add(verstat22.get(i));

robot111.add(robot11.get(i));

robot222.add(robot22.get(i));

robot333.add(robot33.get(i));

}

for(int i = 0;i<n;i++)

{

if(i==(n-1))

{

verstat111.add(verstat11.get(i));

verstat222.add(verstat22.get(i));

robot111.add(robot11.get(0));

robot222.add(robot22.get(0));

robot333.add(robot33.get(0));

}

else

{

verstat111.add(verstat11.get(i));

verstat222.add(verstat22.get(i));

robot111.add(robot11.get(i+1));

robot222.add(robot22.get(i+1));

robot333.add(robot33.get(i+1));

}

}

verstat1 = verstat111;

verstat2 = verstat222;

robot1 = robot111;

robot2 = robot222;

robot3 = robot333;

n\*=2;

System.out.println("Result: " + currentValue);

}

}

public static double[] findBest(int timeModelling, int n, List<Integer> verstat1, List<Integer> verstat2, List<Integer> robot1, List<Integer> robot2,List<Integer> robot3 ) throws ExceptionInvalidNetStructure, ExceptionInvalidTimeDelay

{

int indexBest = 0;

double valueBest = 0;

double[] allValues = new double[n];

for(int i = 0; i < n; i++)

{

ArrayList<PetriSim> list = new ArrayList<PetriSim>();

list.add(new PetriSim(CreateRobots(verstat1.get(i), verstat2.get(i),robot1.get(i),robot2.get(i), robot3.get(i))));

PetriObjModel model = new PetriObjModel(list);

model.setIsProtokol(false);

model.go(timeModelling);

allValues[i] = model.getListObj().get(0).getNet().getListP()[14].getMark()/150000.0;

if(model.getListObj().get(0).getNet().getListP()[14].getMark() > valueBest)

{

indexBest = i;

valueBest = model.getListObj().get(0).getNet().getListP()[14].getMark();

}

}

double[] result = new double[n+1];

result[0] = indexBest;

for(int i = 0; i<n;i++)

{

result[i+1] = allValues[i];

}

return result;

}

public static PetriNet CreateRobots(int verstat1, int verstat2, int robot1, int robot2, int robot3) throws ExceptionInvalidTimeDelay {

ArrayList<PetriP> d\_P = new ArrayList<>();

ArrayList<PetriT> d\_T = new ArrayList<>();

ArrayList<ArcIn> d\_In = new ArrayList<>();

ArrayList<ArcOut> d\_Out = new ArrayList<>();

d\_P.add(new PetriP("P1",1));

d\_P.add(new PetriP("P2",0));

d\_P.add(new PetriP("P3",0));

d\_P.add(new PetriP("P4",0));

d\_P.add(new PetriP("P5",0));

d\_P.add(new PetriP("Р6",verstat1));

d\_P.add(new PetriP("P7",0));

d\_P.add(new PetriP("P8",0));

d\_P.add(new PetriP("P9",0));

d\_P.add(new PetriP("P10",0));

d\_P.add(new PetriP("Р11",verstat2));

d\_P.add(new PetriP("P12",0));

d\_P.add(new PetriP("P13",0));

d\_P.add(new PetriP("P14",0));

d\_P.add(new PetriP("P15",0));

d\_P.add(new PetriP("Р16",robot1));

d\_P.add(new PetriP("Р17",robot2));

d\_P.add(new PetriP("Р18",robot3));

d\_T.add(new PetriT("Надходження",10.0));

d\_T.get(0).setDistribution("exp", d\_T.get(0).getTimeServ());

d\_T.get(0).setParamDeviation(0.0);

d\_T.add(new PetriT("Захоплення",8.0));

d\_T.get(1).setDistribution("unif", d\_T.get(1).getTimeServ());

d\_T.get(1).setParamDeviation(1.0);

d\_T.add(new PetriT("Переміщення до верстату",6.0));

d\_T.add(new PetriT("Вивільнення",8.0));

d\_T.get(3).setDistribution("unif", d\_T.get(3).getTimeServ());

d\_T.get(3).setParamDeviation(1.0);

d\_T.add(new PetriT("Обробка верстатом 1",60.0));

d\_T.get(4).setDistribution("norm", d\_T.get(4).getTimeServ());

d\_T.get(4).setParamDeviation(10.0);

d\_T.add(new PetriT("Захоплення 2",8.0));

d\_T.get(5).setDistribution("unif", d\_T.get(5).getTimeServ());

d\_T.get(5).setParamDeviation(1.0);

d\_T.add(new PetriT("Переміщення до верстату 2",6.0));

d\_T.add(new PetriT("Вивільнення 2",8.0));

d\_T.get(7).setDistribution("unif", d\_T.get(7).getTimeServ());

d\_T.get(7).setParamDeviation(1.0);

d\_T.add(new PetriT("Обробка верстатом 2",60.0));

d\_T.get(8).setDistribution("norm", d\_T.get(8).getTimeServ());

d\_T.get(8).setParamDeviation(10.0);

d\_T.add(new PetriT("Захоплення 3",8.0));

d\_T.get(9).setDistribution("unif", d\_T.get(9).getTimeServ());

d\_T.get(9).setParamDeviation(1.0);

d\_T.add(new PetriT("Переміщення на склад",6.0));

d\_T.add(new PetriT("Вивільнення робота°",8.0));

d\_T.get(11).setDistribution("unif", d\_T.get(11).getTimeServ());

d\_T.get(11).setParamDeviation(1.0);

d\_In.add(new ArcIn(d\_P.get(2),d\_T.get(2),1));

d\_In.add(new ArcIn(d\_P.get(0),d\_T.get(0),1));

d\_In.add(new ArcIn(d\_P.get(7),d\_T.get(6),1));

d\_In.add(new ArcIn(d\_P.get(1),d\_T.get(1),1));

d\_In.add(new ArcIn(d\_P.get(15),d\_T.get(1),1));

d\_In.add(new ArcIn(d\_P.get(3),d\_T.get(3),1));

d\_In.add(new ArcIn(d\_P.get(12),d\_T.get(10),1));

d\_In.add(new ArcIn(d\_P.get(11),d\_T.get(9),1));

d\_In.add(new ArcIn(d\_P.get(17),d\_T.get(9),1));

d\_In.add(new ArcIn(d\_P.get(6),d\_T.get(5),1));

d\_In.add(new ArcIn(d\_P.get(16),d\_T.get(5),1));

d\_In.add(new ArcIn(d\_P.get(9),d\_T.get(8),1));

d\_In.add(new ArcIn(d\_P.get(10),d\_T.get(8),1));

d\_In.add(new ArcIn(d\_P.get(4),d\_T.get(4),1));

d\_In.add(new ArcIn(d\_P.get(5),d\_T.get(4),1));

d\_In.add(new ArcIn(d\_P.get(8),d\_T.get(7),1));

d\_In.add(new ArcIn(d\_P.get(13),d\_T.get(11),1));

d\_Out.add(new ArcOut(d\_T.get(2),d\_P.get(3),1));

d\_Out.add(new ArcOut(d\_T.get(0),d\_P.get(0),1));

d\_Out.add(new ArcOut(d\_T.get(0),d\_P.get(1),1));

d\_Out.add(new ArcOut(d\_T.get(6),d\_P.get(8),1));

d\_Out.add(new ArcOut(d\_T.get(1),d\_P.get(2),1));

d\_Out.add(new ArcOut(d\_T.get(3),d\_P.get(4),1));

d\_Out.add(new ArcOut(d\_T.get(3),d\_P.get(15),1));

d\_Out.add(new ArcOut(d\_T.get(10),d\_P.get(13),1));

d\_Out.add(new ArcOut(d\_T.get(9),d\_P.get(12),1));

d\_Out.add(new ArcOut(d\_T.get(5),d\_P.get(7),1));

d\_Out.add(new ArcOut(d\_T.get(8),d\_P.get(11),1));

d\_Out.add(new ArcOut(d\_T.get(8),d\_P.get(10),1));

d\_Out.add(new ArcOut(d\_T.get(4),d\_P.get(5),1));

d\_Out.add(new ArcOut(d\_T.get(4),d\_P.get(6),1));

d\_Out.add(new ArcOut(d\_T.get(7),d\_P.get(9),1));

d\_Out.add(new ArcOut(d\_T.get(7),d\_P.get(16),1));

d\_Out.add(new ArcOut(d\_T.get(11),d\_P.get(14),1));

d\_Out.add(new ArcOut(d\_T.get(11),d\_P.get(17),1));

PetriNet d\_Net = new PetriNet("graph",d\_P,d\_T,d\_In,d\_Out);

PetriP.initNext();

PetriT.initNext();

ArcIn.initNext();

ArcOut.initNext();

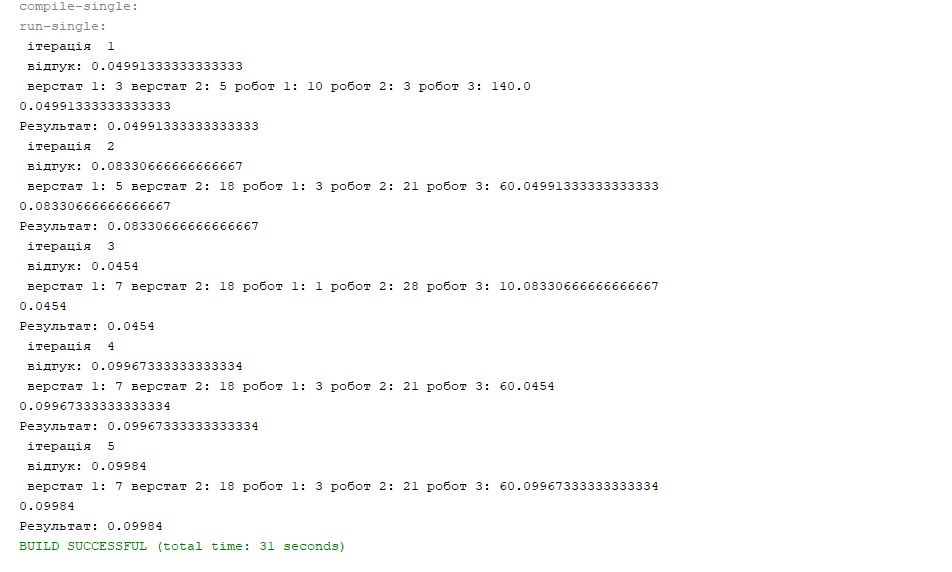
return d\_Net;

}

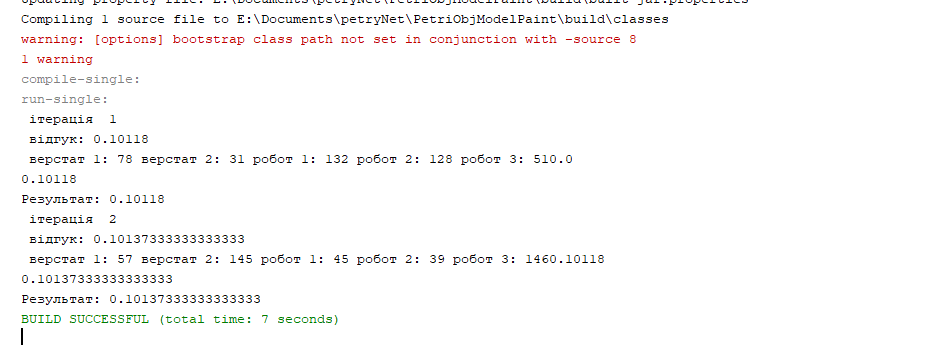
}

1. Виконати пошук оптимальних значень факторів методами еволюційного моделювання для досліджуваної моделі. Повторити пошук при зміні області змінювання факторів.

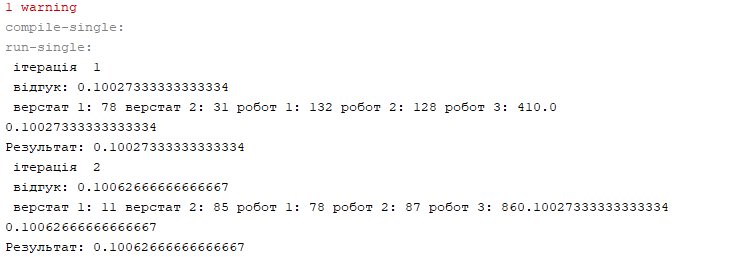
Область 1-7



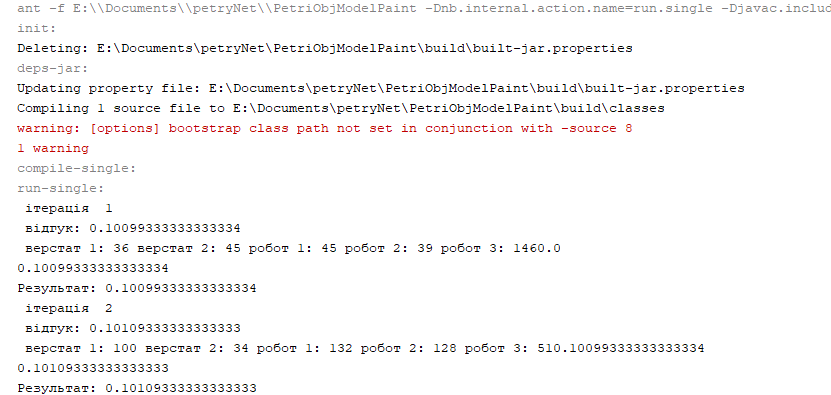
Область 1-150



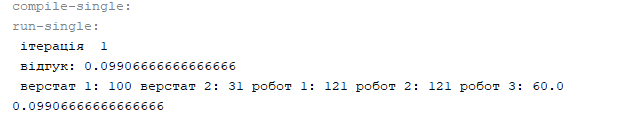
Змінюючи область значень факторів, бачимо, що продуктивність зростає приблизно на 0.01, це означає що для заданої інтенсивності надходження зміна області не сильно повпливала, отже є менша величина при якому система працює не створюючи черг.



12 членів популяції



4 члени популяції



2 члени популяції

чим більша початкова кількість елементів у популяції тим більша вірогідність знайти кращий розв’язок.

1)Можливість змінювати точність пошуку розв’язку.

2)Має можливість покращення розв’язків.

3)Можна змінювати область факторів, алгоритм буде працювати.