НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ СІКОРСЬКОГО”

КАФЕДРА АВТОМАТИЗОВАНИХ СИСТЕМ ОБРОБКИ ІНФОРМАЦІЇ І УПРАВЛІННЯ

Комп’ютерний практикум № 14

з дисципліни

“Моделювання систем”

Виконала:

студентка групи ІС-71

Вознюк О.В.

Перевірила:

старший викладач

Новікова П.А.

Київ-2020

**Отримані результати**

**1**

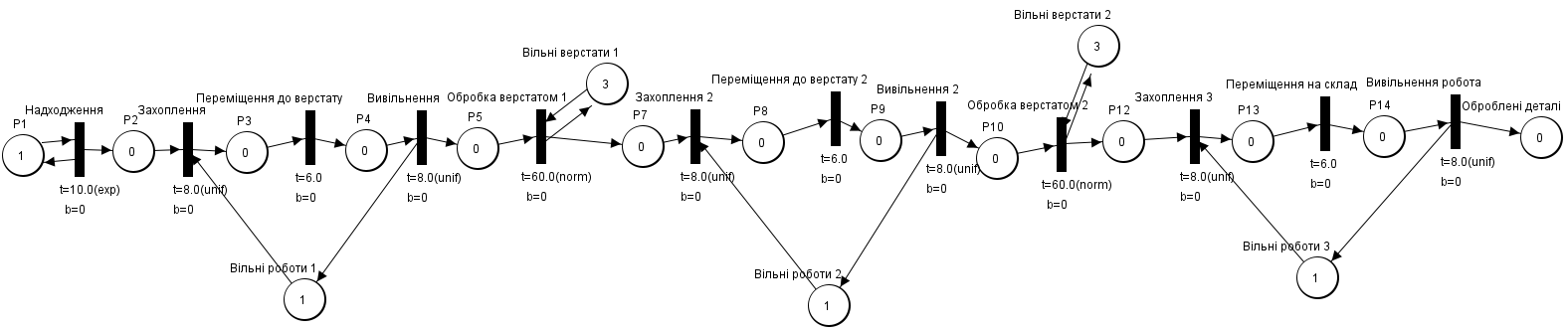


Рисунок 1 – досліджувана схема з 13ого практикуму

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Вільні верстати 1 | Вільні верстати 2 | Вільні роботи 1 | Вільні роботи 2 | Вільні роботи 3 | Значення відгуку моделі (продуктивності) |
| 5 | 16 | 5 | 8 | 1 | 0.03766 |
| 1 | 4 | 3 | 21 | 6 | 0.08418 |
| 31 | 1 | 1 | 1 | 1 | 0.0287 |
| 3 | 5 | 10 | 3 | 14 | 0.04989 |
| 7 | 8 | 2 | 10 | 8 | 0.09076 |
| 13 | 6 | 4 | 5 | 7 | 0.0994 |

Найбільше значення продуктивності у експеримента 6.

**2**

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;

System.out.println("Hello");

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;

}

**3**

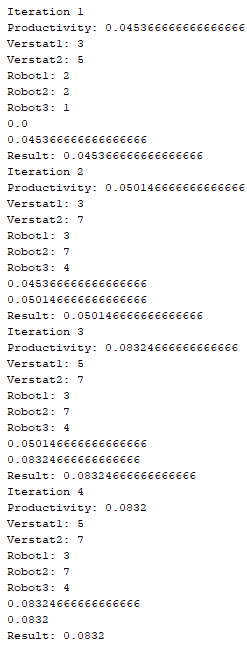


Рисунок 2 - Результати роботи генетичного алгоритму при області змінювання 1-7

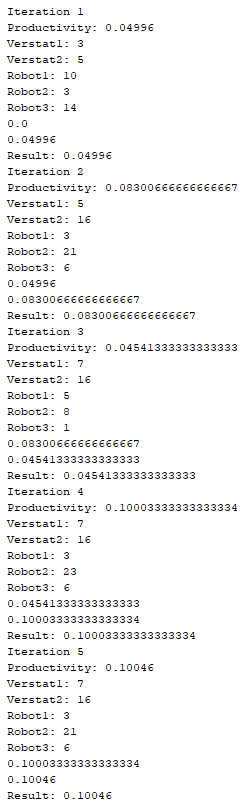


Рисунок 3 - Результати роботи генетичного алгоритму при області змінювання 1-35

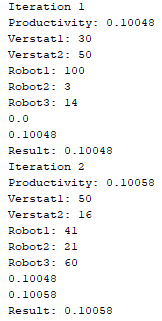


Рисунок 4 - Результати роботи генетичного алгоритму при області змінювання 1-350

Змінюючи область значень факторів, бачимо, що продуктивність не сильно зростає, це означає що для заданої інтенсивності надходження нема потреби так сильно збільшувати кількість верстатів та роботів і є менша величина при якому система працює не створюючи черг.

Кількість ітерацій значно не змінилася, отже на якісність роботи алгоритму дана змінна не впливає.

**4**

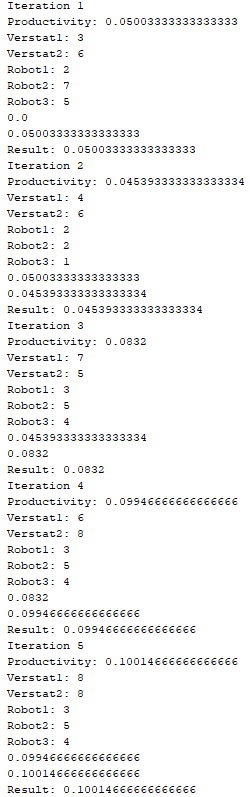


Рисунок 5 – Результати роботи генетичного алгоритму при збільшенні кількості елементів у популяції до 10

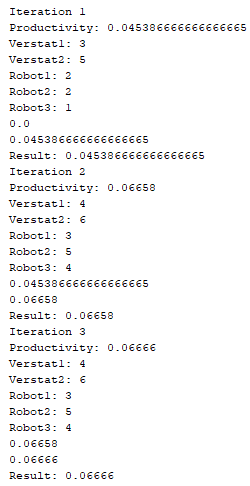


Рисунок 6 – Результати роботи генетичного алгоритму при зменшенні кількості елементів у популяції до 4

Беручи до уваги рисунки 5 та 6 можемо зробити висновок, що чим більша початкова кількість елементів у популяції тим більша вірогідність знайти кращий розв’язок.

**5**

* Дає можливість знайти розв’язок з вказаною точністю.
* Має багато інструментів для покращення поточного значення за допомогою операторів.
* Однаково коректно працює при будь-якій області змінювань факторів.
* Дає кращий результат при більшій кількості початкової популяції