Прізвище: Метельський

Ім’я: Всеволод

Група: КНМ-14

Дата прийняття роботи

у системі Git: 23.04.2017

Дисципліна: Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні

Перевірив: Кривий Р.З.

**Звіт до лабораторної роботи № 4**

**«Реалізація генетичного алгоритму**

**на мові програмування C#»**

**МЕТА РОБОТИ**

Ознайомитися з основними теоретичними відомостями, вивчити еволюційні оператори схрещування та мутації, що використовуються при знаходженні значень цільової функції.

**КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ**

У загальному розумінні генетичні алгоритми (genetic algorithms) – це алгоритми, що використовують механізмами еволюції живої природи – природний відбір і генетичне наслідування. Генетичні алгоритми сьогодні застосовуються в різних галузях. Зокрема їх успішно використовують для розв’язування ряду важливих задач в економіці, бізнесі, техніці. З їх допомогою були розроблені промислові проектні рішення, що сприяли значній економії коштів і ресурсів. Фінансові компанії широко використовують ці засоби для прогнозування розвитку фінансових ринків для управління пакетами цінних паперів.

До основних характеристик ГА належать: розмір популяції (population size), оператор селекції (selection), оператор кросовера (crossover) і правила його використання, оператор мутації (mutation) і його параметри, оператор редукції (reduction), правило (критерій) зупинки процесу виконання генетичного алгоритму (stopping criteria). Оператори селекції, кросовера, мутації і редукції ще називають генетичними операторами.

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

За допомогою засобів мови програмування C# розробити програмне забезпечення для знаходження мінімального та максимального значення цільової функції за допомогою генетичного алгоритму.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ варіанту** | **a** | **b** | **c** | **d** |
| 6 | 50 | -63 | -25 | 1 |

**Код програми:**

using System;

using System.IO;

using System.Windows.Forms;

namespace Genetic\_Algorhitm\_lab4

{

public partial class Form1 : Form

{

public static bool countMax = true;

string path = "test.txt";

public Form1()

{

InitializeComponent();

}

static Random random = new Random();

private void Run\_Click(object sender, EventArgs e)

{

maxChart.Series[0].Points.Clear();

minChart.Series[0].Points.Clear();

lastValuesLabel.Text = String.Empty;

LabelMinCalcs.Text = String.Empty;

countMax = true;

calculate();

countMax = false;

calculate();

File.Delete(path);

}

void calculate()

{

File.Delete(path);

FileStream f = File.Create(path);

f.Close();

Population p;

StreamWriter file = new StreamWriter(path);

int population = 100;

p = new Population(file, population);

int gen = 0;

while (gen <= 1000)

{

p.evolve(file);

++gen;

}

file.Close();

int lineRead = 0;

string[] lines = File.ReadAllLines(path);

foreach (var line in lines)

{

int value = Convert.ToInt32(line.Split(';')[3].Split(' ')[2]);

lineRead++;

if (countMax)

{

if (lineRead < 9)

{

maxChart.Series[0].Points.AddXY(lineRead, value);

lastValuesLabel.Text += line + "\n";

}

else

{

lastValuesLabel.Text += "...\n";

lastValuesLabel.Text += "Max:\n" + lines[lines.Length - 1];

value = Convert.ToInt32(lines[lines.Length - 1].Split(';')[3].Split(' ')[2]);

maxChart.Series[0].Points.AddXY(lineRead, value);

break;

}

}

else

{

if (lineRead < 9)

{

minChart.Series[0].Points.AddXY(lineRead, value);

LabelMinCalcs.Text += line + "\n";

}

else

{

LabelMinCalcs.Text += "...\n";

LabelMinCalcs.Text += "Min:\n" + lines[lines.Length - 1];

value = Convert.ToInt32(lines[lines.Length - 1].Split(';')[3].Split(' ')[2]);

minChart.Series[0].Points.AddXY(lineRead, value);

break;

}

}

}

}

public static double GetRandomNumber(double min, double max)

{

return (random.NextDouble() \* (max - min)) + min;

}

public class Genotype

{

public int[] genes;

public Genotype()

{

this.genes = new int[3];

for (int i = 0; i < genes.Length; i++)

{

this.genes[i] = (int)GetRandomNumber(-10, 53);

}

}

public void mutate()

{

for (int i = 0; i < genes.Length; i++)

{

if (GetRandomNumber(0.0, 100) < 5)

{

this.genes[i] = (int)GetRandomNumber(-10, 53);

}

}

}

}

static Genotype crossover(Genotype a, Genotype b)

{

Genotype c = new Genotype();

for (int i = 0; i < c.genes.Length; i++)

{

if (GetRandomNumber(0.0, 1) < 0.5)

{

c.genes[i] = a.genes[i];

}

else

{

c.genes[i] = b.genes[i];

}

}

return c;

}

public class Phenotype

{

double i\_x;

double i\_x2;

double i\_x3;

public Phenotype(Genotype g)

{

this.i\_x = g.genes[0];

this.i\_x2 = g.genes[1];

this.i\_x3 = g.genes[2];

}

public double evaluate(System.IO.StreamWriter file)

{

double fitness = 0;

if(countMax)

fitness -= 50 - 63 \* i\_x - 25 \* i\_x2 + i\_x3;

else

fitness += 50 - 63 \* i\_x - 25 \* i\_x2 + i\_x3;

file.WriteLine("x: " + i\_x + "; x2: "+i\_x2 + "; x3: "+i\_x3+"; func: "+ fitness);

return fitness;

}

}

public class Individual : IComparable<Individual>

{

public Genotype i\_genotype;

public Phenotype i\_phenotype;

double i\_fitness;

public Individual()

{

this.i\_genotype = new Genotype();

this.i\_phenotype = new Phenotype(i\_genotype);

this.i\_fitness = 0.0;

}

public void evaluate(System.IO.StreamWriter file)

{

this.i\_fitness = i\_phenotype.evaluate(file);

}

int IComparable<Individual>.CompareTo(Individual objI)

{

Individual iToCompare = (Individual)objI;

if (i\_fitness < iToCompare.i\_fitness)

{

return -1;

}

else if (i\_fitness > iToCompare.i\_fitness)

{

return 1;

}

return 0;

}

}

public static Individual breed(Individual a, Individual b)

{

Individual c = new Individual();

c.i\_genotype = crossover(a.i\_genotype, b.i\_genotype);

c.i\_genotype.mutate();

c.i\_phenotype = new Phenotype(c.i\_genotype);

return c;

}

public class Population

{

Individual[] pop;

public Population(System.IO.StreamWriter file, int populationNum)

{

this.pop = new Individual[populationNum];

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

{

this.pop[i] = new Individual();

this.pop[i].evaluate(file);

}

Array.Sort(pop);

}

public void evolve(System.IO.StreamWriter file)

{

Individual a = select(100),

b = select(100),

x = breed(a, b);

if(countMax)

this.pop[0] = x;

else

this.pop[this.pop.Length - 1] = x;

x.evaluate(file);

Array.Sort(pop);

}

Individual select(int popNum)

{

int which = 0;

which = (int)Math.Floor(((float)popNum - 1E-6) \* (1.0 - Math.Pow(GetRandomNumber(0.0, 1.0), 2)));

return pop[which];

}

}

}

}

**Отримані результати:**

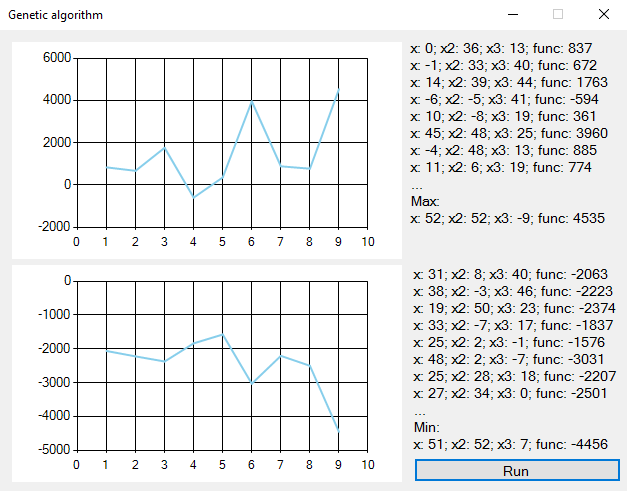


Рис 1. Результат, отриманий в ході першого експерименту

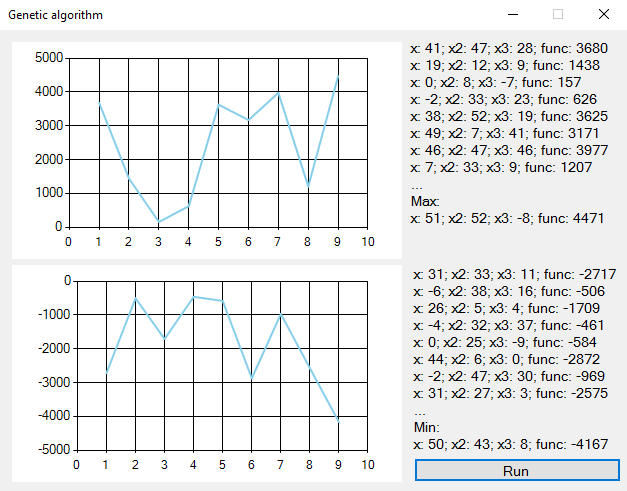


Рис 2. Результат, отриманий в ході другого експерименту

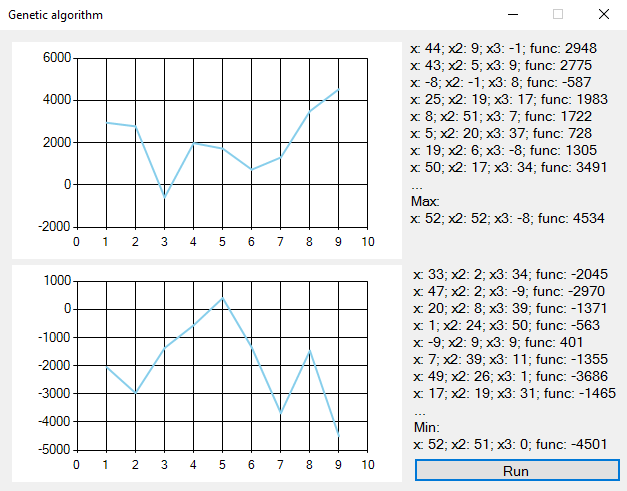


Рис 3. Результат, отриманий в ході третього експерименту

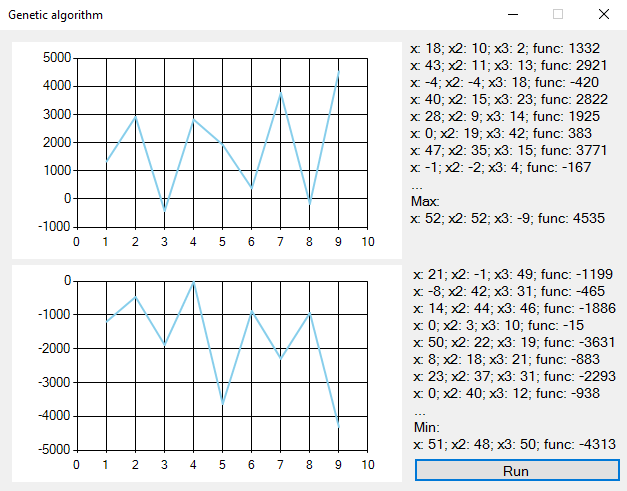


Рис 4. Результат, отриманий в ході четвертого експерименту

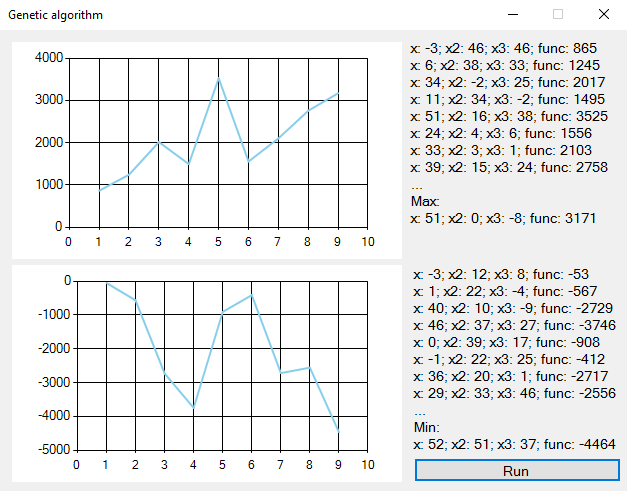


Рис 5. Результат, отриманий в ході п’ятого експерименту

**Висновок:** виконуючи дану лабораторну роботу я, використовуючи засоби мови програмування C# та середовища розробки Visual Studio 2015, розробив програмне забезпечення для визначення мінімуму та максимуму цільової функції за допомогою генетичного алгоритму. Також ознайомився із теоретичними відомостями, основними операторами схрещування та мутації.