Прізвище: Жук

Ім’я: Богдан

Група: КНС-13

Дата прийняття роботи

у системі Git: xx.05.2017

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**

Розробити програму, яка реалізовує генетичний алгоритм пошуку

максимального і мінімального значення цільової функції - f(x) = a + bx +

cx2 + dx3 в інтервалі x = [-10, 53].



8 12 -8 -40 3

**Фрагмент програмного коду:**

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 -= 12 - 8 \* i\_x - 40 \* i\_x2 + 3 \* i\_x3;

else

fitness += 12 - 8 \* i\_x - 40 \* i\_x2 + 3 \* 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;

}

**Результат виконання лабораторного завдання**

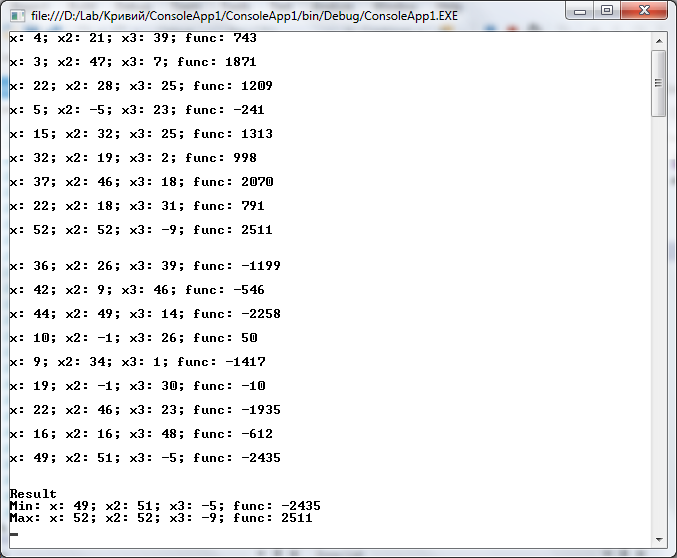


Рис. 1. Результат першого запуску програми

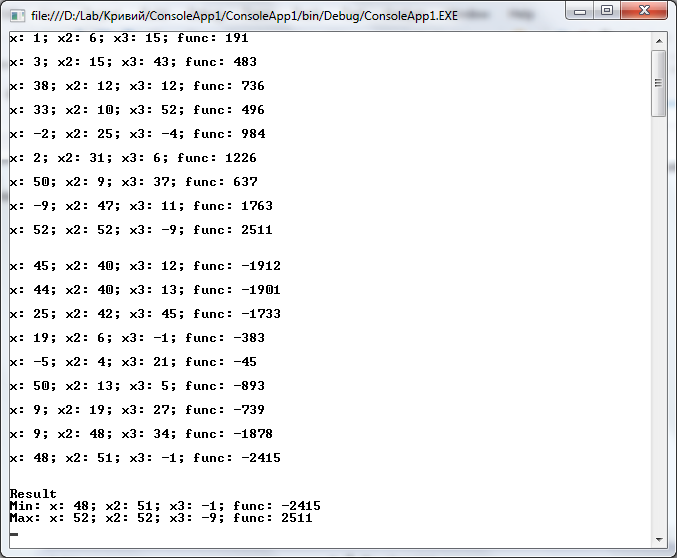


Рис. 2. Результат другого запуску програми

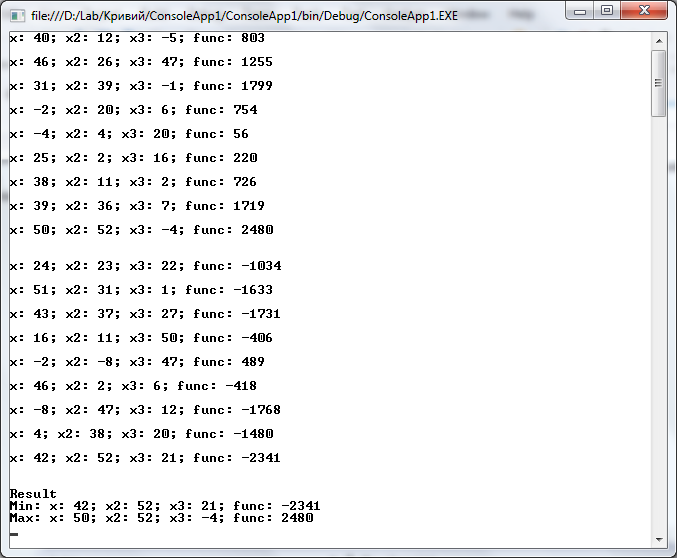


Рис. 3. Результат третього запуску програми

**Висновки**

Під час виконання лабораторної роботи я розробив програму, яка реалізовує генетичний алгоритм пошуку максимального і мінімального значення цільової функції з однією змінною.