|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | Оцінка | підпис |
| КНC-13 | Лабораторна робота №4 | Знаходження екстремумів ф-ії з допомогою генетичного алгоритму |  |  |
| Ракочий Тарас | |
| Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні | | Викладач: | |
| Кривий Р. З. | |

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

**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. Варіант 15**

Розробити програму, яка реалізовує генетичний алгоритм пошуку максимального і мінімального значення цільової функції - f(x) = a + bx + cx2 + dx3 в інтервалі x = [-10, 53].





**Фрагменти коду програми для схрещування та мутації**

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 -= 39 - 96 \* i\_x - 67 \* i\_x2 + 4 \* i\_x3;

else

fitness += 39 - 96 \* i\_x - 67 \* i\_x2 + 4 \* 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];

}

}

**Результат виконання**

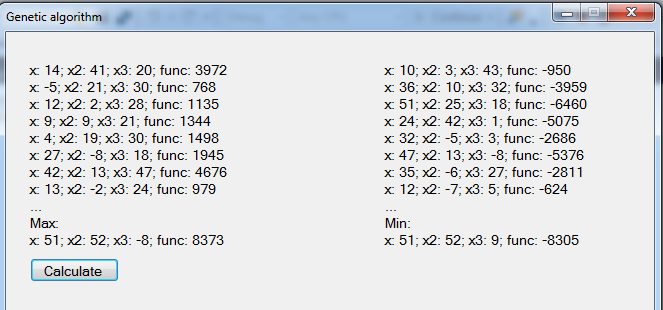


Рис.1. Результат виконання програми.

**Висновок**

Виконуючи лабораторну роботу я ознайомився з основними теоретичними відомостями, вивчив еволюційні оператори схрещування та мутації. Виконав індивідуальне завдання, яка полягає у знаходженні максимуму та мінімуму цільової функції за допомогою генетичного алгоритму.