

**Sztuczna Inteligencja**

**Laboratorium**

**Obliczanie równań przy pomocy algorytmu genetycznego**

**Autor: Krzysztof Wiśniewski 10458**

**Prowadzący: Bartosz Popławski**

**Rok akademicki 2019/20**

**Semestr V**

1. **Założenia projektu**

Celem projektu jest implementacja algorytmu genetycznego w języku C#, z wykorzystaniem Biblioteki AForge.

Program ma mieć zaimplementowane co najmniej 4 równania dwóch zmiennych. Celem działania aplikacji jest znalezienie takich zmiennych x i y, przy pomocy algorytmu genetycznego, aby wynik równania był jak najbliższy zeru.

1. **Funkcje programu**
2. Aplikacja liczy x i y dla wybranego równania by wynik był jak najbliższy zeru.
3. Aplikacja ma zaimplementowane 4 równania :

* x^2y + 8x - xy^2 + 5y
* x^2y+xy^2-2x
* 4x+2y^2+xy
* y^2+2xy^2+3x

1. Aplikacja umożliwia wybranie ilości iteracji
2. Aplikacja umożliwia ustawienie liczebności populacji
3. Aplikacja umożliwia wybór z 3 dostępnych metod selekcji osobników:

* Elite Selection
* Rank Selection
* Roulette Wheel Selection

1. Aplikacja wyświetla wyniki dla wszystkich wykonanych iteracji
2. **Cel ćwiczenia**

Celem wykonanego ćwiczenia jest zaznajomienie się z algorytmem genetycznym oraz poznanie sposobu na jego skuteczną implementacje.

1. **Klasy programu**

Program składa się z dwóch klas głównych

**Klasa Form1**

Klasa odpowiedzialna jest za warstwę wizualną. Jej najważniejszym elementem jest funkcja „genetic” która odpowiada za pobranie parametrów podanych przez użytkownika przy pomocy interfejsu do klasy odpowiedzialnej za algorytm genetyczny, oraz za wyświetlenie zwróconych wyników:

{

int selectedFunction = listBox1.SelectedIndex;

int populationSize = hScrollBar2.Value;

int iterationsNumber = hScrollBar1.Value;

int selectedMethod = listBox2.SelectedIndex;

double[,] result;

GeneticAlgorithm geneticAlgorithm = new GeneticAlgorithm(selectedFunction, populationSize, iterationsNumber, selectedMethod);

result = geneticAlgorithm.returnResult();

int l = result.GetLength(0);

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

{

textBox3.Text = "";

double x = result[i, 0];

double y = result[i, 1];

String text = "";

text += "Iteracja " + (i + 1) + ": ";

text += "x = " + x + " ";

text += "y = " + y + " ";

string[] f = new string[4];

f[0] = "x^2y + 8x - xy^2 + 5y";

f[1] = "x^2y+xy^2-2x";

f[2] = "4x+2y^2+xy";

f[3] = "y^2+2xy^2+3x";

double functionResult;

switch (listBox1.SelectedIndex)

{

case 0:

functionResult = Math.Pow(x, 2) \* y + 8 \* x - x \* Math.Pow(y, 2) + 5 \* y;

break;

case 1:

functionResult = Math.Pow(x, 2) \* y + x \* Math.Pow(y, 2) - 2 \* x;

break;

case 2:

functionResult = 4 \* x + 2 \* Math.Pow(y, 2) + x \* y;

break;

case 3:

functionResult = Math.Pow(y, 2) + 2 \* x \* Math.Pow(y, 2) + 3 \* x;

break;

default:

functionResult = 0;

break;

}

text += f[listBox1.SelectedIndex] + " = " + functionResult + "\r\n";

textBox3.Text += text;

}

}

**Klasa GeneticAlgorithm**

Jest to klasa odpowiedzialna za logikę aplikacji. Jej głównym zadaniem jest przetworzyć dane od użytkownika, przekazane przez klasę Form1, i na ich podstawie zaimplementować algorytm genetyczny i zwrócić wynik. Klasa do obiczeń używa klas AForge.Genetic i AForge.Math.Random Do jej najważniejszych elementów można zaliczyć:

* Konstruktor – To właśnie tu inicjalizowane są obiekty niezbędne do implementacji algorytmu genetycznego ( szablon chromosomu „chromosome” , funkcja przystosowania „fitnessfunction” oraz populacja „population”). Następnie w pętli for wywoływane są kolejne iteracje algorytmu.

public GeneticAlgorithm(int selectedFitnessFunction, int populationSize, int iterationsNumber, int selectedMethod)

{

this.selectedMethod = selectedMethod;

ISelectionMethod selectionMethot = GetSelectionMethod();

DoubleArrayChromosome chromosome = new DoubleArrayChromosome(new AForge.Math.Random.StandardGenerator(), new AForge.Math.Random.StandardGenerator(), new AForge.Math.Random.StandardGenerator(), 2);

FitnessFunction fitnessfunction = new FitnessFunction(selectedFitnessFunction);

Population population = new Population(populationSize, chromosome, fitnessfunction, selectionMethot);

double[,] tmp = new double[iterationsNumber, 2];

this.result = new double[iterationsNumber, 2];

int i;

for (i=0; i < iterationsNumber; i++)

{

population.RunEpoch();

DoubleArrayChromosome bestChromosome = (DoubleArrayChromosome)population.BestChromosome;

double x = (double)bestChromosome.Value.GetValue(0);

double y = (double)bestChromosome.Value.GetValue(1);

tmp[i, 0] = x;

tmp[i, 1] = y;

if(fitnessfunction.Evaluate(bestChromosome)==100)

{

break;

}

}

this.result = new double[i, 2];

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

{

result[j, 0] = tmp[j,0];

result[j, 1] = tmp[j,1];

}

}

* returnResult – metoda zwracająca wartości najlepszych chromosomów dla każdej iteracji.
* FitnessFunction – wewnętrzna funkcja implementująca interfejs IFitnessFunction. Reprezentuje funkcje przystosowania.

private class FitnessFunction : IFitnessFunction

{

int selectedFitnessFunction;

public FitnessFunction(int selectedFitnessFunction)

{

this.selectedFitnessFunction = selectedFitnessFunction;

}

public double Evaluate(IChromosome chromosome)

{

DoubleArrayChromosome actualChromosome = (DoubleArrayChromosome)chromosome;

double x = (double)actualChromosome.Value.GetValue(0);

double y = (double)actualChromosome.Value.GetValue(1);

double functionResult;

switch (selectedFitnessFunction)

{

case 0:

functionResult = Math.Pow(x, 2) \* y + 8 \* x - x \* Math.Pow(y, 2) + 5 \* y;

break;

case 1:

functionResult = Math.Pow(x, 2) \* y + x \* Math.Pow(y, 2) - 2 \* x;

break;

case 2:

functionResult = 4 \* x + 2 \* Math.Pow(y, 2) + x \* y;

break;

case 3:

functionResult = Math.Pow(y, 2) + 2 \* x \* Math.Pow(y, 2) + 3 \* x;

break;

default:

functionResult = 0;

break;

}

functionResult = Math.Abs(functionResult);

double result = 100 - functionResult;

return result;

}

}

* GetSelectionMethod – Metoda zwracająca wybraną metodę selekcji

private ISelectionMethod GetSelectionMethod() //wybor metody selekcji

{

switch (this.selectedMethod)

{

case 0:

return new EliteSelection();

case 1:

return new RankSelection();

case 2:

return new RouletteWheelSelection();

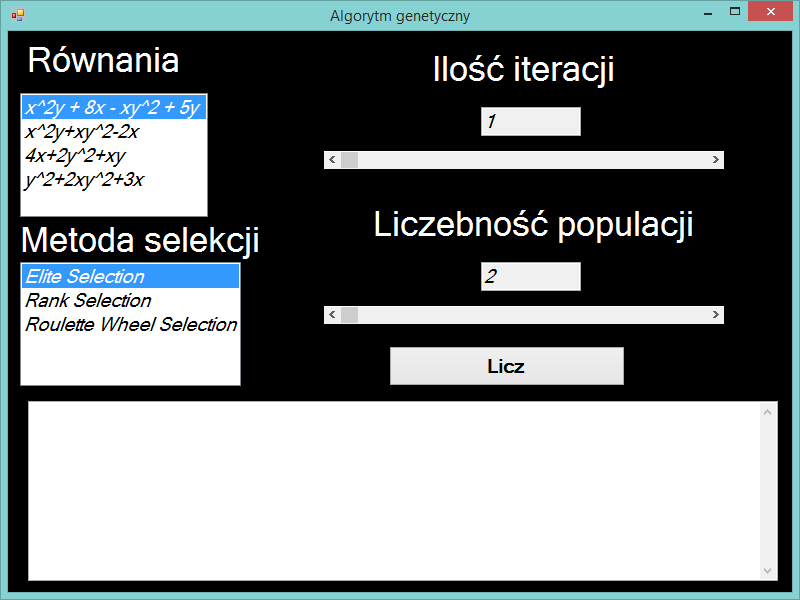
default:

return null;

}

}

1. **Wygląd programu**



Rysunek Główne okno programu

1. **Podsumowanie**

Program spełnia postawione założenia i implementuje algorytm genetyczny.

Pozwala na wyznaczenie X i Y takich które użyte w wybranej funkcji dadzą rezultat jak najbliższy zeru. Jego napisanie pozwoliło mi na praktyczną naukę implementacji algorytmu genetycznego w języku C#.

Aplikacje posiada potencjał do dalszego rozwoju. Kolejnym krokiem mogłoby być wprowadzenie funkcji ręcznego wpisywania równań przez użytkownika. Dobrym pomysłem byłoby również wprowadzenie możliwości zapisu uzyskanych wyników.

1. **Rysunki użyte w sprawozdaniu**
2. Główne okno programu