MINISTRY OF EDUCATION REPUBLIC OF BELARUS
ESTABLISHMENT OF EDUCATION
"BREST STATE TECHNICHNICAL UNIVERSITY"

Practice work №3
«Evolutionary Computation»
Subject: «Lucky Dog Problem with Fithess Sharing Algorithm»

Made by:
Alexey Cherkasov
Checked by:
Pr. Akira Imada

2016

We should:

1. Create a population of N chromosomes of 1000 genes each of whose values is 1, 2, 3 or 4. Chromosome represents 1000 steps of one dog.
2. Assuming fitness being {600 - distance to the nearest sausage}, apply Fitness Sharing Algorithm with sigma= 5
3. Show the table of original and shared fitness in 5 generations, graph of fitness vs generation of highest fitness dogs and routes of highest dogs in 5 generations

20 dogs, 4 sausages

Tables of original and shared fitness in 5 different generations:

Generation 1:

DogNo	Original fitness	Shared fitness
1	584	32,4444444444444
2	580	34,1176470588235
3	588	36,75
4	590	36,875
5	576	38,4
6	562	40,1428571428571
7	562	40,1428571428571
8	564	40,2857142857143
9	570	40,7142857142857
10	566	43,5384615384615
11	574	44,1538461538462
12	576	44,3076923076923
13	578	48,1666666666667
14	580	48,3333333333333
15	562	51,0909090909091
16	566	51,4545454545455
17	554	61,5555555555556
18	558	62
19	538	67,25
20	532	106,4

Generation 2:

DogNo	Original fitness	Shared fitness
1	320	16
2	324	16,2
3	324	16,2
4	324	16,2
5	326	16,3
6	326	16,3
7	328	16,4
8	328	16,4
9	328	16,4
10	328	16,4
11	328	16,4
12	328	16,4
13	328	16,4
14	330	16,5

15	330	16,5
16	330	16,5
17	332	16,6
18	332	16,6
19	332	16,6
20	332	16,6

Generation 100:

Dog№	Original fitness	Shared fitness
1	190	9,5
2	190	9,5
3	192	9,6
4	192	9,6
5	192	9,6
6	194	9,7
7	194	9,7
8	196	9,8
9	196	9,8
10	198	9,9
11	198	9,9
12	198	9,9
13	198	9,9
14	198	9,9
15	198	9,9
16	200	10
17	200	10
18	200	10
19	200	10
20	204	10,2

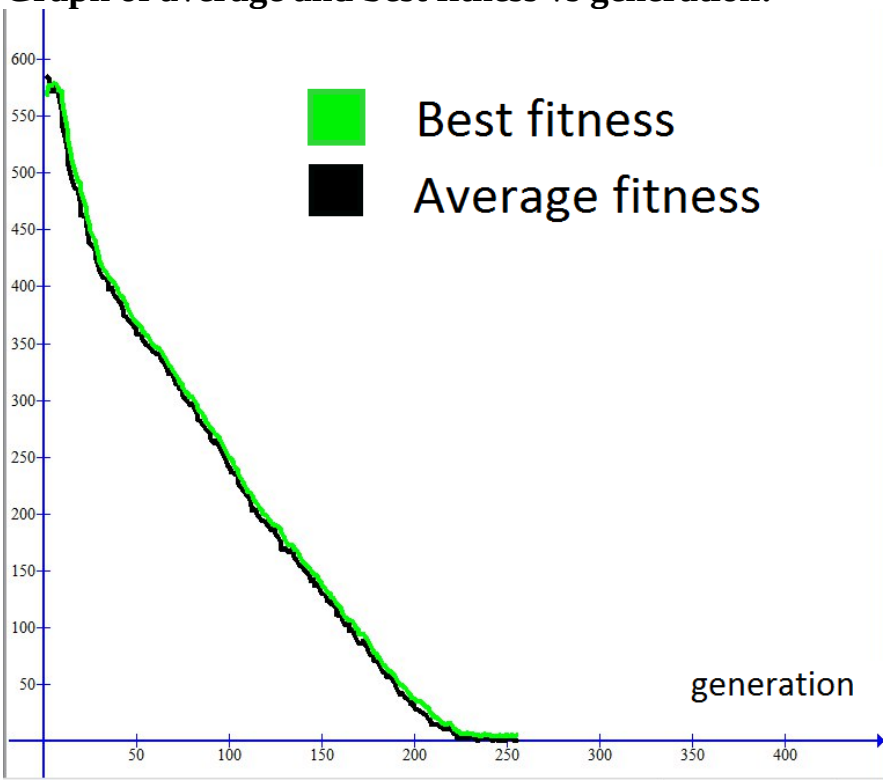
Generation 150:

Dog№	Original fitness	Shared fitness
1	90	4,5
2	90	4,5
3	90	4,5
4	90	4,5
5	92	4,6
6	92	4,6
7	92	4,6
8	92	4,6
9	92	4,6
10	94	4,7
11	94	4,7
12	94	4,7
13	96	4,8
14	96	4,8
15	96	4,8
16	96	4,8
17	98	4,9
18	98	4,9
19	100	5
20	102	5,1

Generation 267:

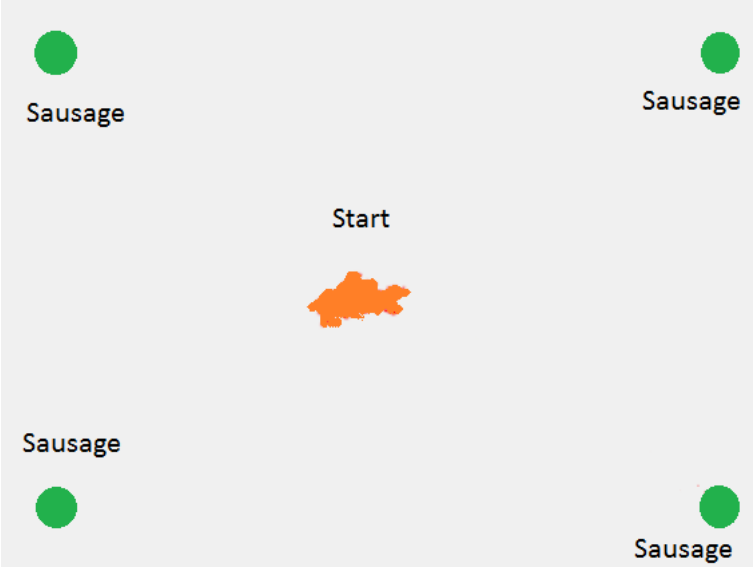
DogNo	Original fitness	Shared fitness
1	0	0
2	2	0,1
3	2	0,1
4	2	0,1
5	2	0,1
6	2	0,1
7	2	0,1
8	2	0,1
9	2	0,1
10	2	0,1
11	2	0,1
12	2	0,1
13	2	0,1
14	4	0,2
15	4	0,2
16	4	0,2
17	6	0,3
18	6	0,3
19	8	0,4
20	10	0,5

Graph of average and best fitness vs generation:



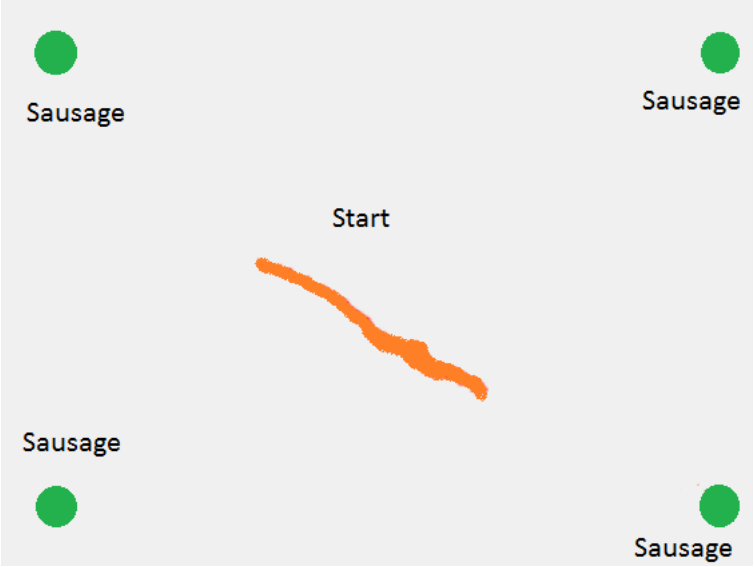
The route of highest fitness dogs in the 1st generations, three intermediate generations, and the final generation:

Generation 1:



Generation 50:

Generation 100:



Generation 150:

Generation 267:

Explanation of the program and results:

In our program we have 20 dogs and 4 sausages, at the last generation all dogs should find a sausage, but the best dog that first find a sausage shares fitness only with nearest dogs, so all dogs go to one direction, to the same sausage, so the task that 20 dogs should go to different sausages can't be executed.

Program Code(C++):

```
#include <iostream>
#include <Windows.h>
#include <vector>
#include <math.h>
#include <fstream>
#include "time.h"

using namespace std;

void NextG(int mas1[20][1000], int mas2[20][1000]);
void Mut(int mas1[20][1000]);
void Fit(int mas1[20][1000], int F[20]);
void OutCord(int mas[20][1000], int m, int c[20][2]);
void Sh_fit_sort(int mas1[20][1000], int c[20][2], int f[20], int F[20]);
void OutBCAF(vector<int> bc, vector<int> af);
int main()
{
	ofstream foutX("Xways.txt"); //making file of X cordinates
	ofstream foutY("Yways.txt"); //making file of Y cordinates
```

```
	srand(time(NULL));
	vector<int> bc;	//
	vector<int> af;
	int ch[20][1000];
	int child[20][1000];
	int F[20];
	int f[20];				//massive shared fitness of chromosom
	int c[20][2];				//finish cordinates

	int y1 = 0, g = 1;

	for (int i = 0; i < 20; i++)  //input 1-st generation
	{
		for (int f = 0; f < 1000; f++)
		{
			ch[i][f] = rand() % 4 + 1;
		}
	}
	Fit(ch, F);
	Sh_fit_sort(ch, c, f, F);
	OutCord(ch, 0, c);
	bc.push_back(f[0]);	//calculating best and average fitness of generation
	y1 = 0;
	for (int i = 0; i < 20; i++)
	{
		y1 = y1 + f[i];
	}
	af.push_back(y1 / 20);
	cout << bc[0] << ' ';	//display fitnes of best chromosom

	for (int end = 0; end == 0;)
	{
		//-------------------------------------------------------------------------1
		NextG(ch, child);
		Mut(child);
		Fit(child, F);
		Sh_fit_sort(child, c, f, F);
		OutCord(child, g, c);
		bc.push_back(f[0]);	//calculating best and average fitness of generation 1
		y1 = 0;
		for (int i = 0; i < 20; i++)
		{
			y1 = y1 + f[i];
		}
		af.push_back(y1 / 20);
		cout << bc[g] << ' ';	//display fitnes of best chromosom 1
		if (bc[g] == 0) end = 1; //stoping 1
		g++;
		//-----------------------------------------------------------------------------2
		if (end == 0) {
			NextG(child, ch);
			Mut(ch);
			Fit(ch, F);
			Sh_fit_sort(ch, c, f, F);
			OutCord(ch, g, c);
			bc.push_back(f[0]);	//calculating best and average fitness of generation 2
			y1 = 0;
			for (int i = 0; i < 20; i++)
			{
				y1 = y1 + f[i];
			}
			af.push_back(y1 / 20);
			cout << bc[g] << ' ';	//display fitnes of best chromosom 2
			if (bc[g] == 0) end = 1;  //stoping 2
			g++;
		}
	}
	OutBCAF(bc, af);

	std::cout << "(To see more information, check text files: BC.txt, AF.txt, GEN.txt, Xways.txt, Yways.txt)" << endl << endl;
	foutX << endl;
	foutY << endl;
	system("pause");
	return 0;
}

void NextG(int mas1[20][1000], int mas2[20][1000])
{
	srand(time(NULL));
	int  x1, x2, e;
	for (int s = 0; s < 20; s++)
	{
```

```
                x1 = rand() % 10;
                x2 = rand() % 10;
                for (int i = 0; i < 1000; i++)  //uniform crossover
                {
                        e = rand() % 2;
                        if (e == 1) { mas2[s][i] = mas1[x1][i]; }
                        else { mas2[s][i] = mas1[x2][i]; }
                }
        }
}

void Mut(int mas1[20][1000])
{
        srand(time(NULL));
        for (int i = 0; i < 20; i++)
        {
                for (int f = 0; f < 1000; f++)
                {
                        int mut = rand() % 20;
                        if (mut == 2) {
                                mas1[i][f] = rand() % 4 + 1;
                        }
                }
        }
}

void Fit(int mas1[20][1000], int F[20])
{
        int X1 = 500, Y1 = 500, x1 = 0, y1 = 0, x2 = 0, y2 = 0, x3 = 0, y3 = 0, x4 = 0, y4 = 0, f1, f2, f3, f4;
        for (int i = 0; i < 20; i++)
        {
                F[i] = 1;
                X1 = 500;
                Y1 = 500;
                for (int f = 0; f < 1000; f++)   // dog is moving 1
                {
                        switch (mas1[i][f])
                        {
                        case 1: { Y1 = Y1 + 1; break; }
                        case 2: { Y1 = Y1 - 1; break; }
                        case 3: { X1 = X1 + 1; break; }
                        case 4: { X1 = X1 - 1; break; }
                        default: cout << "error" << endl;
                        }
                        if (X1 == 1001) { X1 = 0; }
                        if (X1 == -1) { X1 = 1000; }
                        if (Y1 == 1001) { Y1 = 0; }
                        if (Y1 == -1) { Y1 = 1000; }
                        if (X1 == 200 && Y1 == 200) { F[i] = 0; }   // if dog find sausage 1
                        if (X1 == 200 && Y1 == 800) { F[i] = 0; }   // if dog find sausage 2
                        if (X1 == 800 && Y1 == 800) { F[i] = 0; }   // if dog find sausage 3
                        if (X1 == 800 && Y1 == 200) { F[i] = 0; }   // if dog find sausage 4
                }
                if (F[i] = !0)
                {
                        x1 = 200 - X1;
                        y1 = 200 - Y1;
                        x2 = 200 - X1;
                        y2 = 800 - Y1;
                        x3 = 800 - X1;
                        y3 = 800 - Y1;
                        x4 = 800 - X1;
                        y4 = 200 - Y1;
                        f1 = fabs(x1) + fabs(y1);
                        f2 = fabs(x2) + fabs(y2);
                        f3 = fabs(x3) + fabs(y3);
                        f4 = fabs(x4) + fabs(y4);
                        if (f1 <= f2 && f1 <= f3 && f1 <= f4) F[i] = f1;
                        if (f2 <= f1 && f2 <= f3 && f2 <= f4) F[i] = f2;
                        if (f3 <= f2 && f3 <= f1 && f3 <= f4) F[i] = f3;
                        if (f4 <= f2 && f4 <= f3 && f4 <= f1) F[i] = f4;
                }
        }
}

void OutCord(int mas[20][1000], int m, int c[20][2])
{
        ofstream foutX("Xways.txt", ios_base::app); //making file of X cordinates
        ofstream foutY("Yways.txt", ios_base::app); //making file of Y cordinates
        int X1 = 500, Y1 = 500;
        foutX << "-" << m + 1 << "----------------------------" << endl;
        foutY << "-" << m + 1 << "----------------------------" << endl;
```

```
	for (int i = 0; i < 20; i++)
	{
		foutX << i + 1 << "////" << endl;
		foutY << i + 1 << "////" << endl;
		X1 = 500;
		Y1 = 500;

		for (int f = 0; f < 1000; f++)   // dog is moving 1
		{
			switch (mas[i][f])
			{
			case 1: { Y1 = Y1 + 1; break; }
			case 2: { Y1 = Y1 - 1; break; }
			case 3: { X1 = X1 + 1; break; }
			case 4: { X1 = X1 - 1; break; }
			default: cout << "error" << endl;
			}
			foutX << X1 << endl;
			foutY << Y1 << endl;
			c[i][1] = X1;
			c[i][2] = Y1;
		}
		foutX << endl;
		foutY << endl;
	}
}
```