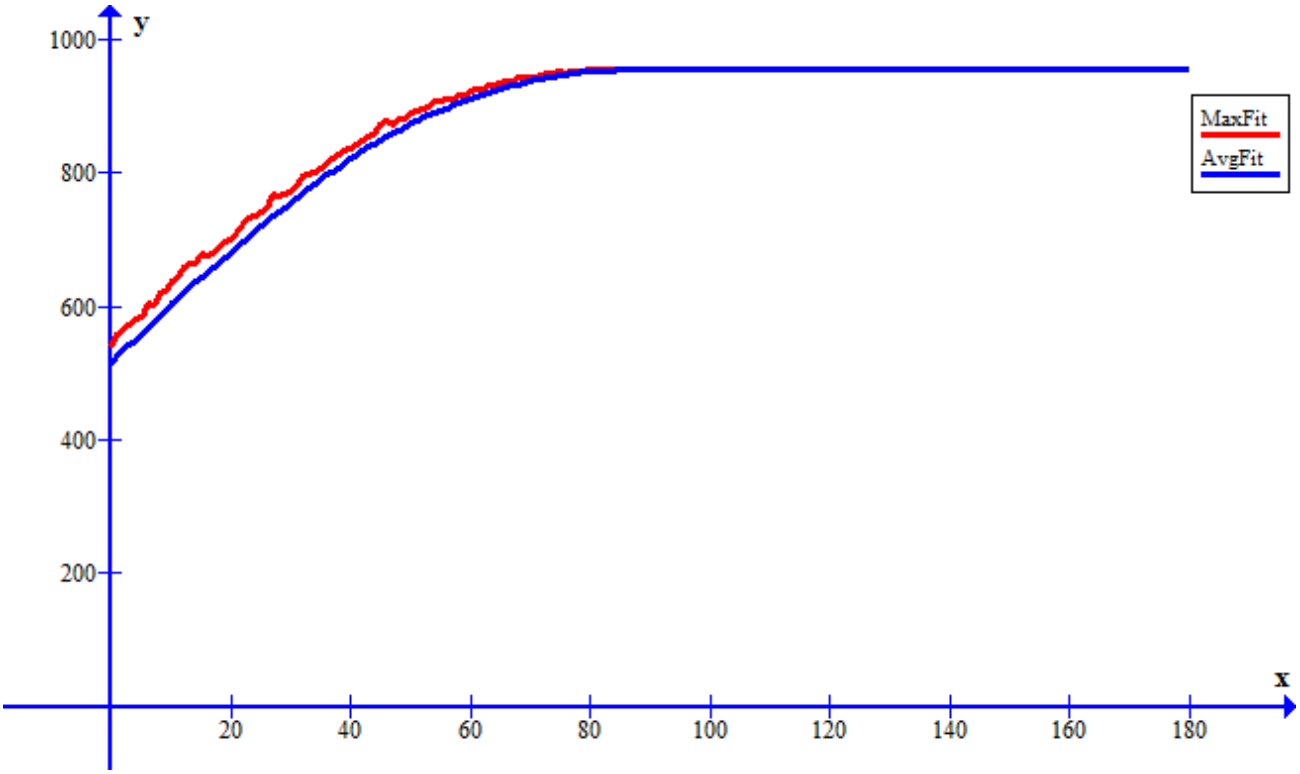
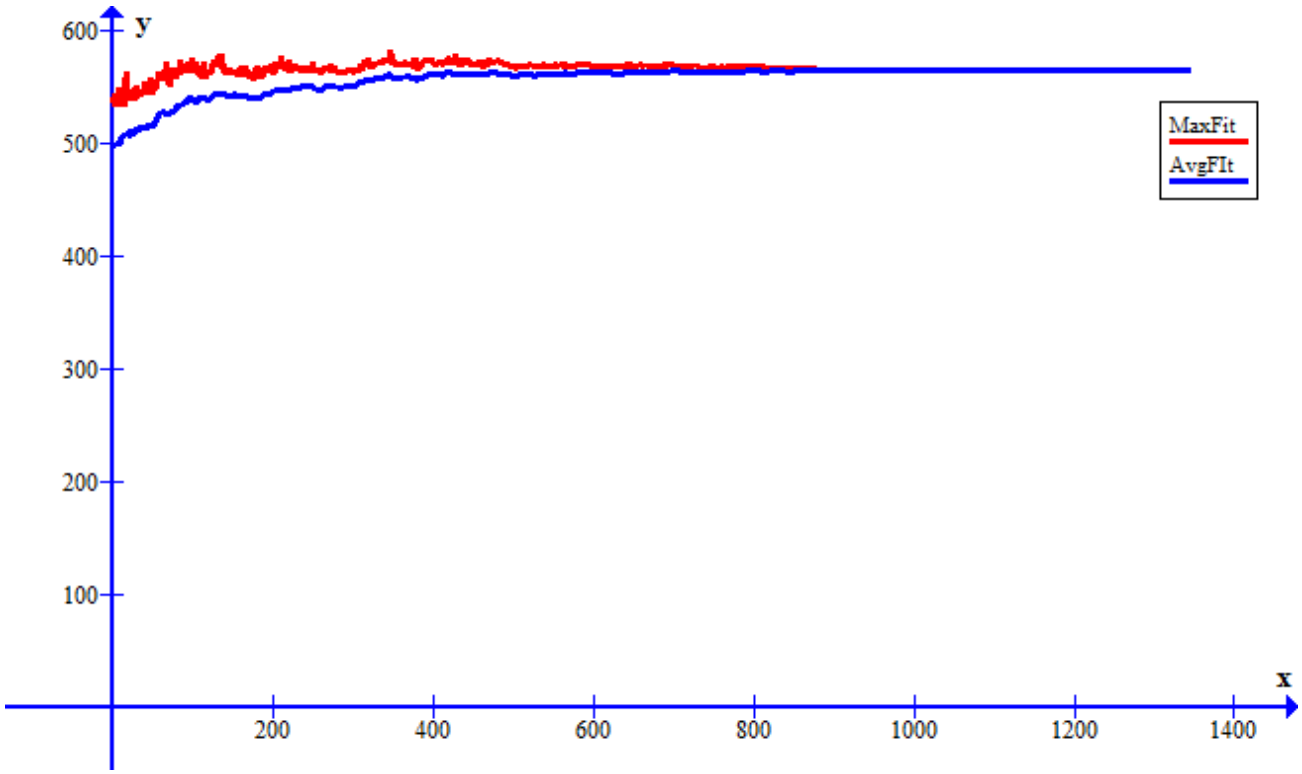
1. Create 100 random binary-chromosomes each with 1000 genes.
2. Fitness is the number of “1” in one chromosome – the more the better.
3. Select 2 chromosomes at random from the better half of the population.
4. Create a child chromosome by a uniform crossover.
5. Repeat from 2. to 4. 100 times and create the next generation.
6. Repeat 5. until the fitness value does not change any more.
7. Show the result:
(1) Desplay the best chromosome in the 1st, an intermediate & final generation.
(2) Desplay the best and average fitness vs. generation.

Truncate selection



Roulette selection



Selected table

First generation		Middle generation		Last generation	
Number	how many times is selected	Number	how many times is selected	Number	how many times is selected
0	2	0	2	0	2
1	3	1	3	1	2
2	1	2	2	2	2
3	2	3	1	3	2
4	3	4	2	4	2
5	1	5	3	5	2
6	3	6	2	6	2
7	1	7	1	7	1
8	2	8	2	8	2
9	0	9	2	9	2
10	4	10	3	10	2
11	2	11	2	11	2
12	0	12	1	12	1
13	3	13	3	13	2
14	3	14	2	14	3
15	1	15	1	15	3
16	2	16	1	16	1
17	2	17	3	17	1
18	2	18	1	18	2

19	2	19	2	19	2
20	2	20	2	20	1
21	2	21	2	21	2
22	3	22	3	22	2
23	2	23	3	23	3
24	1	24	1	24	2
25	1	25	2	25	1
26	3	26	1	26	3
27	1	27	1	27	2
28	2	28	2	28	2
29	2	29	2	29	1
30	2	30	1	30	2
31	2	31	3	31	3
32	2	32	2	32	3
33	1	33	2	33	2
34	3	34	3	34	1
35	2	35	2	35	3
36	1	36	1	36	2
37	2	37	3	37	2
38	2	38	2	38	2
39	3	39	2	39	3
40	3	40	3	40	1
41	2	41	3	41	2
42	1	42	1	42	2
43	3	43	2	43	2
44	2	44	2	44	2
45	3	45	2	45	2
46	2	46	2	46	2
47	1	47	1	47	2
48	2	48	1	48	3
49	2	49	2	49	1
50	1	50	2	50	3
51	3	51	1	51	2
52	3	52	2	52	1
53	2	53	3	53	3
54	2	54	1	54	1
55	2	55	2	55	2
56	2	56	1	56	3
57	3	57	3	57	2
58	3	58	2	58	1
59	2	59	1	59	1
60	0	60	1	60	2
61	3	61	2	61	2
62	3	62	3	62	3
63	0	63	3	63	2

64	2	64	1	64	1
65	3	65	3	65	2
66	1	66	2	66	2
67	2	67	2	67	2
68	3	68	2	68	2
69	0	69	3	69	2
70	3	70	2	70	1
71	2	71	2	71	3
72	0	72	1	72	2
73	2	73	2	73	2
74	4	74	3	74	3
75	0	75	3	75	3
76	2	76	2	76	1
77	3	77	2	77	2
78	1	78	2	78	2
79	3	79	2	79	2
80	3	80	3	80	2
81	1	81	3	81	2
82	3	82	1	82	2
83	3	83	2	83	2
84	0	84	2	84	3
85	3	85	2	85	2
86	4	86	2	86	2
87	0	87	2	87	3
88	3	88	2	88	1
89	2	89	3	89	3
90	1	90	1	90	1
91	3	91	1	91	2
92	3	92	3	92	2
93	1	93	3	93	2
94	1	94	2	94	1
95	2	95	1	95	2
96	0	96	2	96	2
97	4	97	2	97	2
98	3	98	3	98	3
99	1	99	1	99	2

Source code

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace siit_1
{
    class generation
    {
        List<int[]> gens;
        List<int> fitness { get; }
        List<float> probability { get; }
        List<int> chromSelect;
        public float averagefitness = 0f;

        int rando = 0;

        public generation()
        {
            gens = new List<int[]>();
            fitness = new List<int>();
            probability = new List<float>();
            chromSelect = new List<int>();

            for (int j = 0; j < 100; j++)
            {
                int[] gen = new int[1000];
                gens.Add(gen);
                fitness.Add(0);
                probability.Add(0f);
                chromSelect.Add(0);
            }
        }

        public generation(List<int[]> new_gens)
        {
            gens = new List<int[]>();
            fitness = new List<int>();
            probability = new List<float>();
            chromSelect = new List<int>();
            gens = new_gens;
            for (int j = 0; j < 100; j++)
            {
                fitness.Add(0);
                probability.Add(0f);
                chromSelect.Add(0);
            }
        }

        public void randomize()
        {
            Random rand = new Random();
            for (int i = 0; i < 100; i++)
            {
                for (int j = 0; j < 1000; j++)
                {
```

```
            gens[i][j] = rand.Next() % 2;
        }
    }
}
public void setFitness()
{
    for (int i = 0; i < 100; i++)
    {
        for (int j = 0; j < 1000; j++)
        {
            if (gens[i][j] == 1) fitness[i] += 1;
        }
    }
}
public void setProbability()
{
    int mass = 0; ;
    for (int i = 0; i < 100; i++)
    {
        mass += fitness[i];
    }
    averagefitness = mass / 100;
    for (int i = 0; i < 100; i++)
    {
        probability[i] = (float)fitness[i] / (float)mass;
    }
}
public int[] newChild()
{

    Random rand = new Random(DateTime.Now.TimeOfDay.Milliseconds + rando);
    rando++;
    if (rando == 10000000) rando = 0;
    int rand_num = rand.Next(1000000000);
    float sum = 0f;
    int[] chrom_1 = new int[1000], chrom_2 = new int[1000];

       for(int i = 0; i < 100; i++)
       {
           sum += probability[i]* 1000000000;
           if (rand_num <= sum)
           {
               chromSelect[i]++;
               chrom_1 = gens[i];
               break;
           }

       }
    //chrom_1 = gens[rand_num];
    sum = 0f;
    rand_num = rand.Next(1000000000);
    for (int i = 0; i < 100; i++)
    {
        sum += probability[i] * 1000000000;
        if (rand_num <= sum)
        {
            chromSelect[i]++;
            chrom_2 = gens[i];
            break;
        }
    }
```

```
            //chrom_2 = gens[rand_num];


            //Random lamb = new Random(rand.Next());
            int[] new_chrom = new int[1000];
            for (int i = 0; i < 1000; i++)
            {
                if (rand.Next() % 2 == 1) new_chrom[i] = chrom_1[i];
                else new_chrom[i] = chrom_2[i];
            }
            return new_chrom;
        }
        public int bestFitness()
        {
            return fitness.Max();
        }

        public void Sort()
        {
            for (int i = 0; i < 100 - 1; i++)
            {
                bool swapped = false;
                for (int j = 0; j < 100 - i - 1; j++)
                {
                    if (fitness[j] < fitness[j + 1])
                    {
                        int[] tmp_gen = gens[j];
                        gens[j] = gens[j + 1];
                        gens[j + 1] = tmp_gen;

                        int tmp_fit = fitness[j];
                        fitness[j] = fitness[j + 1];
                        fitness[j + 1] = tmp_fit;
                        swapped = true;
                    }

                }
                if (!swapped) break;
            }
        }
        public float getAverageFit()
        {
            return averagefitness;
        }
        public void WriteTable(StreamWriter file1,StreamWriter file2)
        {
            for (int i = 0; i < 100; i++) {
                file1.WriteLine(chromSelect[i].ToString());
                file2.WriteLine(i.ToString());
                }
            file1.WriteLine();
            file1.WriteLine();
        }
    }
}


using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.IO;

namespace siit_1
{
    class Program
    {
        static void Main(string[] args)
        {
            StreamWriter avgFitFile = new StreamWriter("averageFit.txt");
            StreamWriter maxFitFile = new StreamWriter("maxFit.txt");
            StreamWriter numGenFile = new StreamWriter("numGen.txt");
            StreamWriter tableFile = new StreamWriter("Table.txt");
            StreamWriter tablenum = new StreamWriter("Num.txt");
            generation old_gens = new generation();
            old_gens.randomize();
            old_gens.setFitness();
            old_gens.setProbability();
            int maxFit = 0;
            int numGeneration = 0;
            for (int j = 0; j < 10000; numGeneration++)
            {
                numGenFile.WriteLine(numGeneration.ToString());
                Console.WriteLine(old_gens.bestFitness() + " " + old_gens.getAverageFit());
                if (old_gens.bestFitness() == 1000) break;
                List<int[]> new_tmp = new List<int[]>();
                //old_gens.Sort();
                for (int i = 0; i < 100; i++)
                {
                    new_tmp.Add(old_gens.newChild());
                }
                old_gens.WriteTable(tableFile,tablenum);
                generation new_gens = new generation(new_tmp);
                old_gens = new_gens;
                old_gens.setFitness();
                old_gens.setProbability();
                avgFitFile.WriteLine(old_gens.getAverageFit().ToString());
                maxFitFile.WriteLine(old_gens.bestFitness().ToString());
                if (old_gens.bestFitness() > maxFit)
                {
                    maxFit = old_gens.bestFitness();
                    j = 0;
                }
                else j++;

            }
            tablenum.Close();
            tableFile.Close();
            numGenFile.Close();
            avgFitFile.Close();
            maxFitFile.Close();
        }
    }
}
```