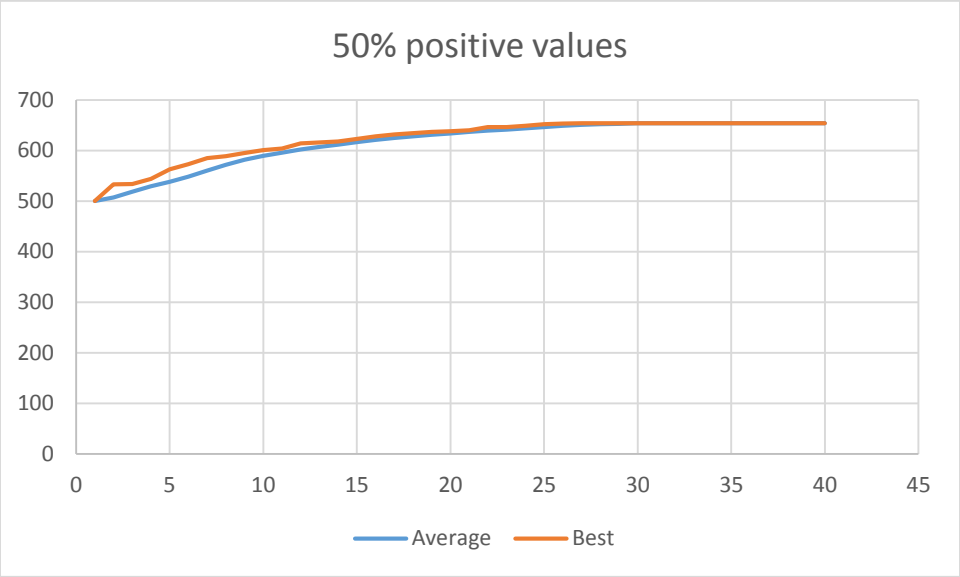
1) 50% positive values:

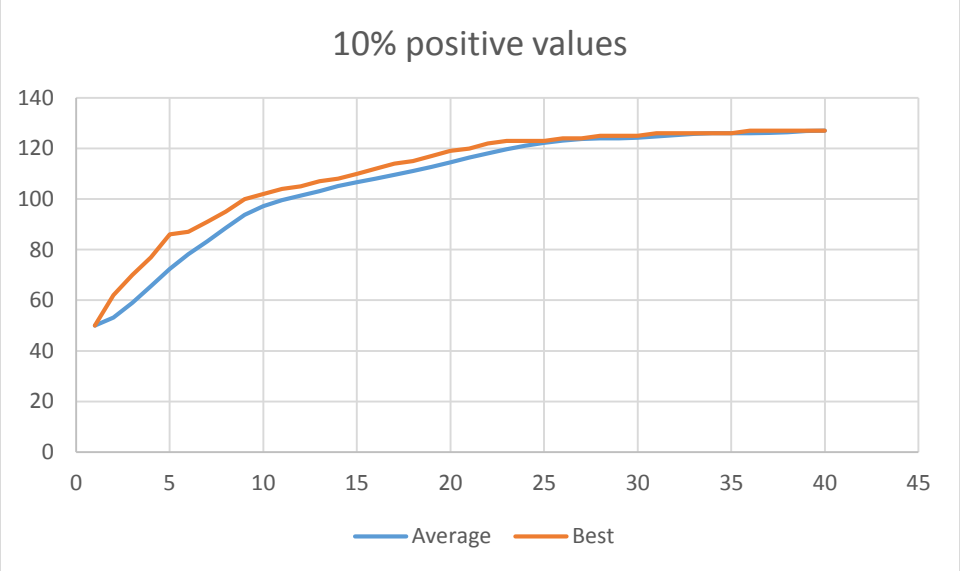


Iterations data:

iteration	average	best
1	500	500
2	507.36	533
3	518.38	534
4	529.16	544
5	538.23	563
6	548.37	573
7	560.32	585
8	571.86	589
9	581.67	595
10	589.3	601
11	595.95	604
12	602.21	614
13	606.84	616
14	611.72	618
15	616.51	623
16	620.93	628
17	624.51	632
18	628.22	634
19	630.81	637
20	633.82	638
21	636.52	640
22	639.1	646
23	641.47	646

24	643.77	649
25	646.05	652
26	648.53	653
27	650.51	654
28	651.93	654
29	652.78	654
30	653.62	654
31	654	654
32	654	654
33	654	654
34	654	654
35	654	654
36	654	654
37	654	654
38	654	654
39	654	654
40	654	654

2) 10% positive values:

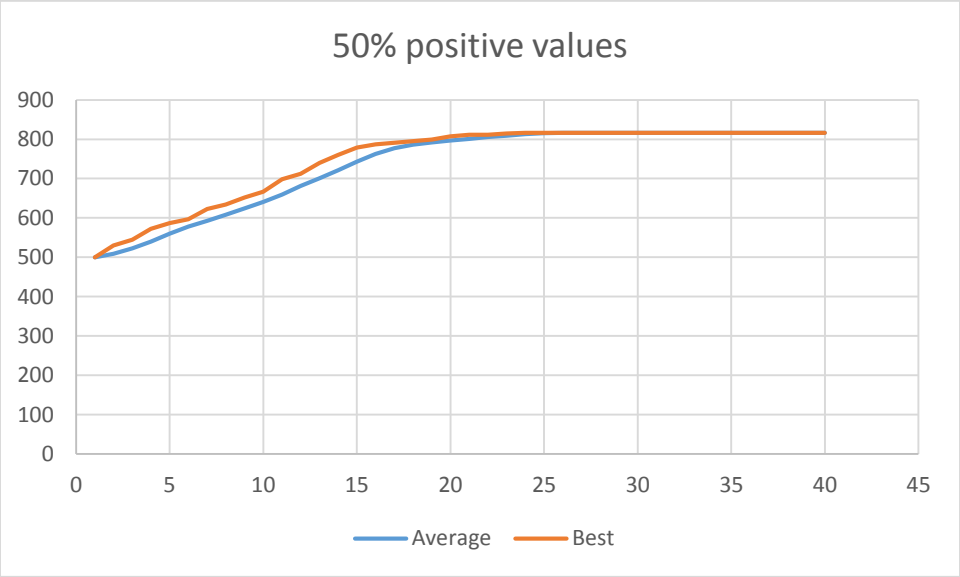


iteration	average	best
1	50	50
2	53.21	62
3	58.96	70
4	65.62	77

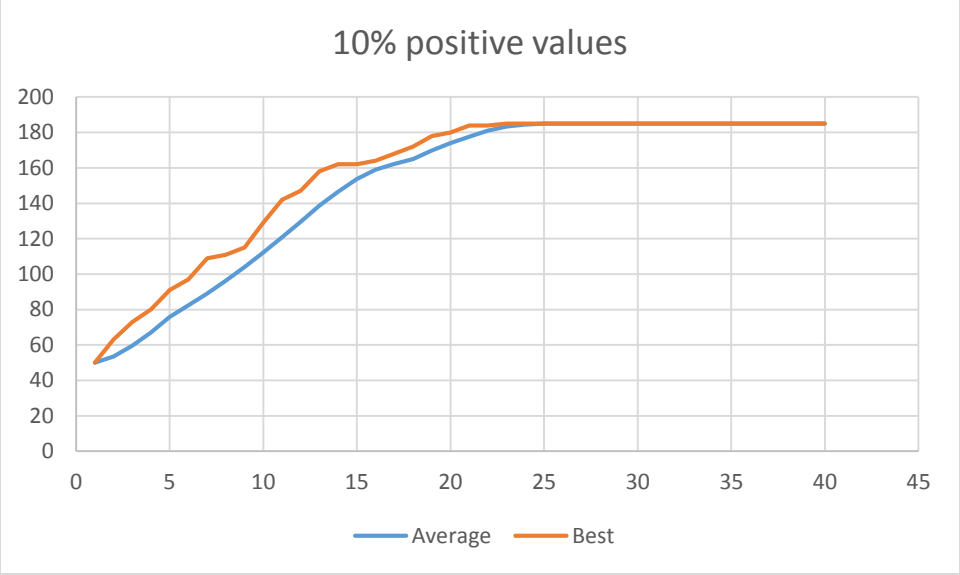
5	72.45	86
6	78.17	87
7	83.31	91
8	88.54	95
9	93.76	100
10	97.22	102
11	99.56	104
12	101.4	105
13	103.13	107
14	105.12	108
15	106.7	110
16	108.09	112
17	109.6	114
18	111.03	115
19	112.68	117
20	114.55	119
21	116.37	120
22	118	122
23	119.67	123
24	121.14	123
25	122.19	123
26	123.14	124
27	123.73	124
28	124.01	125
29	124.05	125
30	124.24	125
31	124.75	126
32	125.24	126
33	125.82	126
34	126	126
35	126	126
36	126.01	127
37	126.1	127
38	126.37	127
39	126.88	127
40	127	127

Also I tried to improve results changing the degree of library function which I use to sort gens in chromosomes in initial generation. The best results in this case looks like:

1) 50% positive values:



2) 10% positive values:



Code:

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

namespace GeneticSystem
{
```

```
public class GeneticSystem
{
	private readonly int _chromosomeSize;
	private readonly int _generationSize;
	private readonly Random _random = new Random();

	private List<bool[]> _currentGeneration;

	public GeneticSystem(int chromosomeSize, int generationSize)
	{
		_chromosomeSize = chromosomeSize;
		_generationSize = generationSize;
		_currentGeneration = new List<bool[]>();
	}

 public List<bool[]> CurrentGeneration
 {
     get
     {
         return _currentGeneration;
     }
 }

	public double AverageFitness
	{
		get { return _currentGeneration.Average(item => GetFitness(item)); }
	}

	public int MaxFitness
	{
		get { return _currentGeneration.Max(item => GetFitness(item)); }
	}

	public List<bool[]> CreateFirstGeneration(int maxPositiveCount)
	{
	    for (int i = 0; i < _generationSize; i++)
		{
			bool[] chromosome = new bool[_chromosomeSize];
			for (int j = 0; j < maxPositiveCount; j++)
			{
		        chromosome[j] = true;
			}

		    chromosome = chromosome.OrderBy(item => _random.Next()).ToArray();
			_currentGeneration.Add(chromosome);
		}

	    return _currentGeneration;
	}

	public List<bool[]> CreateNextGeneration()
	{
		List<bool[]> bestChromosomes =
_currentGeneration.OrderByDescending(GetFitness)
			.Take(_generationSize / 2)
			.ToList();

		_currentGeneration.Clear();
```

```
            int firstIndex, secondIndex, attemps;
                    for (int i = 0; i < _generationSize; i++)
                    {
                attemps = 0;
                do
                {
                    firstIndex = _random.Next(_generationSize / 2);
                    secondIndex = _random.Next(_generationSize / 2);
                } while (GetDifferencesCount(bestChromosomes[firstIndex],
bestChromosomes[secondIndex]) == 0 && ++attemps < 10);


                _currentGeneration.Add(CrossChromosomes(bestChromosomes[firstIndex],
bestChromosomes[secondIndex], true));
                    }

            return _currentGeneration;
              }

              public int GetFitness(bool[] chromosome)
              {
                    return chromosome.Where(i => i).Count();
              }

        public int GetDifferencesCount(bool[] first, bool[] second)
        {
            int count = 0;
            for (int i = 0; i < first.Length; i++)
            {
                if (first[i] != second[i])
                {
                    count++;
                }
            }

            return count;
        }

              public bool[] CrossChromosomes(bool[] firstParent, bool[] secondParent,
bool useRandomPositionToCross = true)
              {
            int position = useRandomPositionToCross ? _random.Next(1, _chromosomeSize) :
_chromosomeSize / 2;
                    bool[] firstChild = new bool[_chromosomeSize];
                    bool[] secondChild = new bool[_chromosomeSize];
                    for (int i = 0; i < _chromosomeSize; i++)
                    {
                           if (i < position)
                           {
                                  firstChild[i] = firstParent[i];
                                  secondChild[i] = secondParent[i];
                           }
                           else
                           {
                                  firstChild[i] = secondParent[i];
                                  secondChild[i] = firstParent[i];
                           }
                    }
```

```
                    return GetFitness(firstChild) > GetFitness(secondChild) ? firstChild
: secondChild;
                }
        }
}
```

```
using GeneticSystem;
using System.IO;

namespace GeneticSystemConsole
{
        class Program
        {
                static void Main(string[] args)
                {
                        GeneticSystem.GeneticSystem geneticSystem = new
GeneticSystem.GeneticSystem(1000, 100);
                        geneticSystem.CreateFirstGeneration(50);

                        int limitCount = 0;
              StreamWriter averageDataFile = new StreamWriter("average.txt");
              StreamWriter bestDataFile = new StreamWriter("best.txt");
                        while (limitCount++ < 100)
                        {
                   Console.WriteLine("======================================");
                   Console.WriteLine(geneticSystem.AverageFitness + " " +
geneticSystem.MaxFitness);
                  averageDataFile.WriteLine(geneticSystem.AverageFitness);
                  bestDataFile.WriteLine(geneticSystem.MaxFitness);
                                geneticSystem.CreateNextGeneration();
                        }

              averageDataFile.Close();
              bestDataFile.Close();
                        Console.ReadKey();
                }
        }
}
```