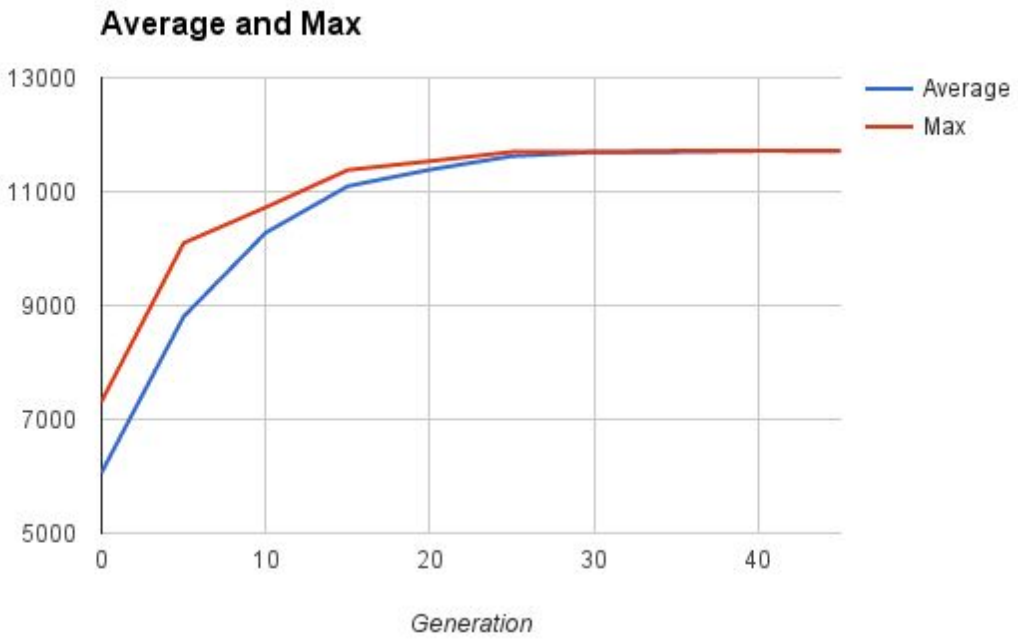
10/27 - Semenyuk Zhenya

Limits are:

Bag size limit = 10000
Amount of each item limit = 100

Item	Price	Volume
1	6	4
2	3	2
3	6	7
4	6	2
5	10	4
6	10	2
7	7	5
8	10	10
9	10	1
10	10	3
11	4	2
12	2	9
13	6	5
14	5	8
15	6	1
16	7	2
17	3	7
18	10	7
19	4	5
20	1	7

Generation	Average	Max
0	6067,175	7317
5	8805	10097
10	10279,1	10724
15	11096,225	11380
20	11385,525	11535
25	11623,825	11698
30	11694,65	11698
35	11696,6	11716
40	11716	11716
45	11712,6	11716



item (number)	Chromosome 1 (amount of items)	Chromosome 2 (amount of items)	Chromosome 3 (amount of items)	Chromosome 4 (amount of items)	Chromosome 5 (amount of items)

Generation 0

item (number)	Chromosome 1	Chromosome 2	Chromosome 3	Chromosome 4	Chromosome 5
1	14	9	72	5	95
2	88	97	21	32	28
3	34	30	52	46	61
4	47	47	98	39	61

5	93	56	73	50	61
6	55	25	55	98	18
7	94	73	35	61	33
8	76	70	12	29	76
9	61	45	70	12	42
10	54	99	77	91	46
11	87	91	69	69	68
12	35	78	71	37	67
13	99	94	4	87	50
14	69	22	51	87	69
15	26	30	74	67	96
16	47	70	32	42	67
17	3	70	37	1	92
18	20	68	90	84	49
19	82	41	67	65	19
20	56	89	23	96	33

Generation 15

item (number)	Chromosome 1	Chromosome 2	Chromosome 3	Chromosome 4	Chromosome 5
1	95	95	95	95	95
2	86	86	86	32	32
3	83	83	83	28	83
4	95	68	80	95	95
5	98	93	93	93	93
6	93	93	99	99	99
7	99	97	94	97	97
8	99	99	99	99	99
9	94	94	94	94	94
10	95	93	95	95	95
11	87	87	87	87	87
12	45	67	45	45	45
13	89	89	89	89	89
14	69	96	69	76	76
15	96	96	96	96	96
16	88	88	88	88	88
17	82	82	82	82	82

18	92	92	32	92	92
19	67	67	67	67	67
20	58	58	52	58	58

Generation 25

item (number)	Chromosome 1	Chromosome 2	Chromosome 3	Chromosome 4	Chromosome 5
1	95	95	95	95	95
2	86	86	86	86	86
3	95	95	95	95	95
4	95	95	95	95	95
5	93	93	93	93	93
6	99	99	99	99	99
7	99	99	99	99	99
8	99	99	99	99	99
9	94	94	94	94	94
10	95	94	95	95	93
11	87	87	87	87	87
12	67	67	67	67	67
13	96	96	96	96	96
14	99	99	96	96	99
15	96	96	96	96	96
16	88	88	88	88	88
17	82	82	82	82	82
18	92	92	92	92	92
19	67	67	67	67	67
20	58	58	58	58	58

Generation 35

item (number)	Chromosome 1	Chromosome 2	Chromosome 3	Chromosome 4	Chromosome 5
1	95	95	95	95	95
2	86	86	86	86	86
3	95	95	95	95	95
4	95	95	95	95	95
5	93	93	93	93	93

6	99	99	99	99	99
7	99	99	99	99	99
8	99	99	99	99	99
9	94	94	94	94	94
10	95	95	95	95	95
11	87	87	87	87	87
12	67	67	67	67	67
13	99	99	99	99	99
14	99	99	99	99	99
15	96	96	96	96	96
16	88	88	88	88	88
17	82	82	82	82	82
18	92	92	92	92	92
19	67	67	67	67	67
20	58	58	58	58	58

Generation 45

item (number)	Chromosome 1	Chromosome 2	Chromosome 3	Chromosome 4	Chromosome 5
1	95	95	95	95	95
2	86	86	86	86	86
3	95	95	95	95	95
4	95	95	95	95	95
5	93	93	93	93	93
6	99	99	99	99	99
7	99	99	99	99	99
8	99	99	99	99	99
9	94	94	94	94	94
10	95	95	95	95	95
11	87	87	87	87	87
12	67	67	67	67	67
13	99	99	99	99	99
14	99	99	99	99	99
15	96	96	96	96	96
16	88	88	88	88	88
17	82	82	82	82	82

18	92	92	92	92	92
19	67	67	67	67	67
20	58	58	58	58	58

Source code

Constants.cs

```
namespace trip
{
   public static class Constants
   {
      public const int GENES_NUMBER = 20;
      public const int CHROMOSOMES_NUMBER = 40;
      public const int BAG_SIZE = 10000;
      public const int AMOUNT_LIMIT = 100;
   }
}
```

Chromosome.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static trip.Constants;

namespace trip
{
   public class Item
   {
      public int Price { get; set; }
      public int Volume { get; set; }

      public Item(Random rnd)
      {
         Volume = rnd.Next(10) + 1;
         Price = rnd.Next(10) + 1;
      }

      public static List<Item> Get(Random rnd)
      {
```

```
        var items = new List<Item>();

        for (int i = 0; i < GENES_NUMBER; i++)
            items.Add(new Item(rnd));

        return items;
    }
}

public class Chromosome : IComparable<Chromosome>
{
    public List<Item> items { get; set; }
    public List<int> Amounts { get; set; }
    public int BagVolume { get; set; }
    public int BagPrice { get; set; }


    public Chromosome(Random rnd, List<Item> bag)
    {
        items = bag;
        Amounts = new List<int>();
        int limit = AMOUNT_LIMIT,//BAG_SIZE / items.Min(x => x.Volume),
            i;

        for (i = 0; i < GENES_NUMBER; i++)
            Amounts.Add(rnd.Next(limit));

        CalculateBagVolumeAndPrice();
    }

    public void CalculateBagVolumeAndPrice()
    {
        BagVolume = 0;
        BagPrice = 0;

        for (int i = 0; i < GENES_NUMBER; i++)
        {
            BagVolume += items.ElementAt(i).Volume *
                Amounts.ElementAt(i);
            BagPrice += items.ElementAt(i).Price *
                Amounts.ElementAt(i);
        }
    }

    public int CompareTo(Chromosome compareChromosome)
    {
```

```
            if (compareChromosome == null)
                return -1;
            else
                return compareChromosome.BagPrice.CompareTo(BagPrice);
        }

        public Chromosome CreateChild(Chromosome p2, Random rnd)
        {
            int ch;
            Chromosome child = new Chromosome(rnd, new List<Item>(items));
            child.Amounts.Clear();

            for (int j = 0; j < GENES_NUMBER; j++)
            {
                ch = rnd.Next(2);
                if(ch == 1)
                {
                    child.Amounts.Add(Amounts.ElementAt(j));
                }
                else
                {
                    child.Amounts.Add(p2.Amounts.ElementAt(j));
                }

            }

            //mutation
            for (int j = 0; j < GENES_NUMBER; j++)
            {
                ch = rnd.Next(20);
                if (ch == 10)
                {
                    child.Amounts.Insert(j, child.Amounts.ElementAt(j) + rnd.Next(AMOUNT_LIMIT)
% AMOUNT_LIMIT);
                    child.Amounts.Remove(child.Amounts.ElementAt(j));

                }
            }
            child.CalculateBagVolumeAndPrice();
            return child;
        }
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static trip.Constants;

namespace trip
{
    class Program
    {
        static void Main(string[] args)
        {
            Random rnd = new Random(DateTime.Now.Millisecond);
            var chromosomes = new List<Chromosome>();
            var childs = new List<Chromosome>();
           // Item mock = new Item(rnd);
            var items = Item.Get(rnd);
            Console.WriteLine("Item's price and volume");
            foreach(var item in items)
            {
                Console.Write("Price = $" + item.Price + ", Volume = " + item.Volume + " ");
            }
            Console.WriteLine();
            int i = 0, j;
            int generation = 0;
            Chromosome p1, p2;

            for (j = 0; j < CHROMOSOMES_NUMBER; j++)
            {
                var chr = new Chromosome(rnd, items);
                if (chr.BagVolume < BAG_SIZE)
                    chromosomes.Add(chr);
                else j--;
            }

            double averageFitness;

            while (generation < 50)
            {

                chromosomes.Sort();

                if(generation % 5 == 0)
                {
                    Console.WriteLine("----------------------------------------");
```

```
            Console.WriteLine("Generation: " + generation);
            for (int k = 0; k < 5; k++)
            {
                var chr = chromosomes.ElementAt(k);
                Console.WriteLine("$" + chr.BagPrice + " - V:" + chr.BagVolume);
                for (int g = 0; g < GENES_NUMBER; g++)
                    Console.Write(chr.Amounts.ElementAt(g) + " ");
                Console.WriteLine();
            }
            averageFitness = chromosomes.Average(x => x.BagPrice);
            Console.WriteLine(averageFitness);
        }


        for (j = 0; j < CHROMOSOMES_NUMBER; j++)
        {
            p1 = chromosomes.ElementAt(rnd.Next(CHROMOSOMES_NUMBER / 2));
            p2 = chromosomes.ElementAt(rnd.Next(CHROMOSOMES_NUMBER / 2));
            var child = p1.CreateChild(p2, rnd);

            if (child.BagVolume < BAG_SIZE)
                childs.Add(child);
            else j--;
        }

        chromosomes.Clear();
        chromosomes.AddRange(childs);
        childs.Clear();

        generation++;
    }
  }
 }
}
```