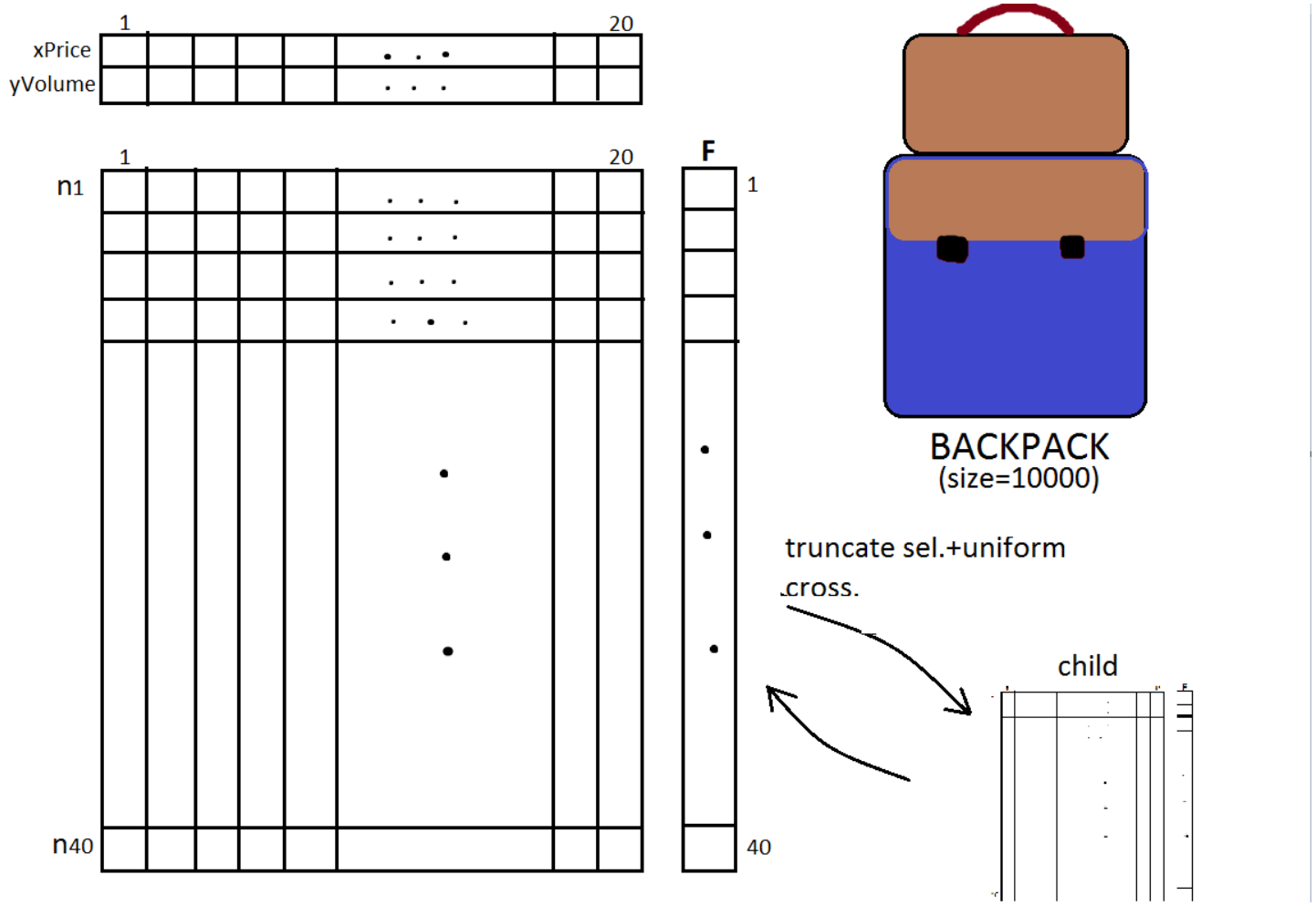
Knapsack Problem

Task:

20 items in the store with price and size of each. We have backpack in which you need to put a certain amount of each item. We need to get maximum price of items from our backpack without overflow.

Description:



xPrice – price of each item;
yVolume - size of each item;
BACKPACK size = 10000;
Meaning (n1,1…) = random from 1 to 100;
sumPrice – full price of chromosome;
sumVolume – full size of chromosome;
F(fitness) = sumPrice (if sumVolume > backpack size than F=0);

Making next generations by truncate selection and uniform crossover.

Code:

Class1.cs

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

namespace лаба_6
{
  class TableOfParam
  {
    public Random rnd = new Random();
    public int[] xPrice = new int [20];
    public int[] yVolume = new int [20];
    public TableOfParam()
    {
      for (int i = 0; i < 20; i++)
      {
        xPrice[i] = rnd.Next(1, 11);
        Console.Write(xPrice[i] + "  ");
      }
      Console.WriteLine();
      for (int i = 0; i < 20; i++)
      {
        yVolume[i] = rnd.Next(1, 11);
        Console.Write(yVolume[i] + "  ");
      }
    }
  }

  class Class1
  {
    TableOfParam tB = new TableOfParam();
    Random rnd = new Random();
    const int gen = 20;
    const int chromosome = 40;
    const int sizeOfBag = 10000;
    int[,] population = new int [chromosome, gen];
    int[,] newPopulation = new int[chromosome, gen];
    int[] fitness = new int[chromosome];
    int[] keys = new int[chromosome];
    public Class1()
    {
      //создание поколения
      int sumPrice;
      int sumVolume;
      for (int i = 0; i < chromosome; i++)
      {
        sumPrice = 0;
        sumVolume = 0;
        while (fitness[i] == 0)
        {
          for (int j = 0; j < gen; j++)
          {

            population[i, j] = rnd.Next(1, 101);
            sumPrice += (tB.xPrice[j] * population[i, j]);
            sumVolume += (tB.yVolume[j] * population[i, j]);
          }
          //считаем фитнесы первого поколения
          fitness[i] = (sumVolume > sizeOfBag) ? 0 : sumPrice;
        }
      }
      newPopulation = population;
    }
```

```
    public void Cross()
    {
      //кроссинговер
      for (int i = 0; i < chromosome; i++)
      {
        int a = rnd.Next(0, chromosome / 2);
        int b = rnd.Next(0, chromosome / 2);
        for (int j = 0; j < gen; j++)
        {
          population[i, j] = (rnd.Next(0, 2) == 0) ? newPopulation[a, j] :
newPopulation[b, j];
        }
      }
    }

    public void Mutation()
    {
      int sumPrice;
      int sumVolume;
      for (int i = 0; i < chromosome; i++)
      {
        sumPrice = 0;
        sumVolume = 0;
        for (int j = 0; j < gen; j++)
        {
          int value = rnd.Next(0, gen);
          population[i, j] = (value == 0) ? rnd.Next(1, 101) : population[i, j];
          sumPrice += (tB.xPrice[j] * population[i, j]);
          sumVolume += (tB.yVolume[j] * population[i, j]);
        }
        fitness[i] = (sumVolume > sizeOfBag) ? 0 : sumPrice;
      }
      //сортировка
      for (int i = 0; i < chromosome; i++)
      {
        keys[i] = i;
      }
      Array.Sort(fitness, keys);
      for (int i = 0, n = chromosome - 1; i < chromosome; i++, n--)
      {
        for (int j = 0; j < gen; j++)
        {
          newPopulation[i, j] = population[keys[n], j];
        }
      }
    }

    public int GetMax()
    {
      return fitness.Max();
    }
    public int GetAverage()
    {
      return (int)fitness.Average();
    }

    public int[,] GetPopulation()
    {
      return newPopulation;
    }
    public int[,] GetFirstPopulation()
    {
      return population;
    }
```

```
  }
}
```

Program.cs

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

namespace лаба_6
{
  class Program
  {
    static void Main(string[] args)
    {
      List<int[,]> populations = new List<int[,]>();
      Class1 cl = new Class1();
      populations.Add(cl.GetFirstPopulation());
      for (int j = 0; j < 20; j++)
      {
        using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@"FirstChrom.txt", true))
        {
          file.WriteLine(populations[0][0, j]);
        }
      }
      int i = -1;
      while (i < 100)
      {
        i++;
        cl.Cross();
        cl.Mutation();
        populations.Add(cl.GetPopulation());
        using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"Maximum.txt",
true))
        {
          file.WriteLine(cl.GetMax());
        }
        using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"Average.txt",
true))
        {
          file.WriteLine(cl.GetAverage());
        }
        using (System.IO.StreamWriter file = new
System.IO.StreamWriter(@"Iterations.txt", true))
        {
          file.WriteLine(i);
        }
      }
      for (int j = 0; j < 20; j++)
      {
        using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"M1Chrom.txt",
true))
        {
          file.WriteLine(populations[25][0, j]);
        }
        using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"M2Chrom.txt",
true))
        {
          file.WriteLine(populations[50][0, j]);
        }
        using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"M3Chrom.txt",
true))
        {
          file.WriteLine(populations[75][0, j]);
        }
```

```
                using (System.IO.StreamWriter file = new System.IO.StreamWriter(@"LastChrom.txt",
true))
                {
                  file.WriteLine(populations[i][0, j]);
                }
              }
            }
          }
        }
```

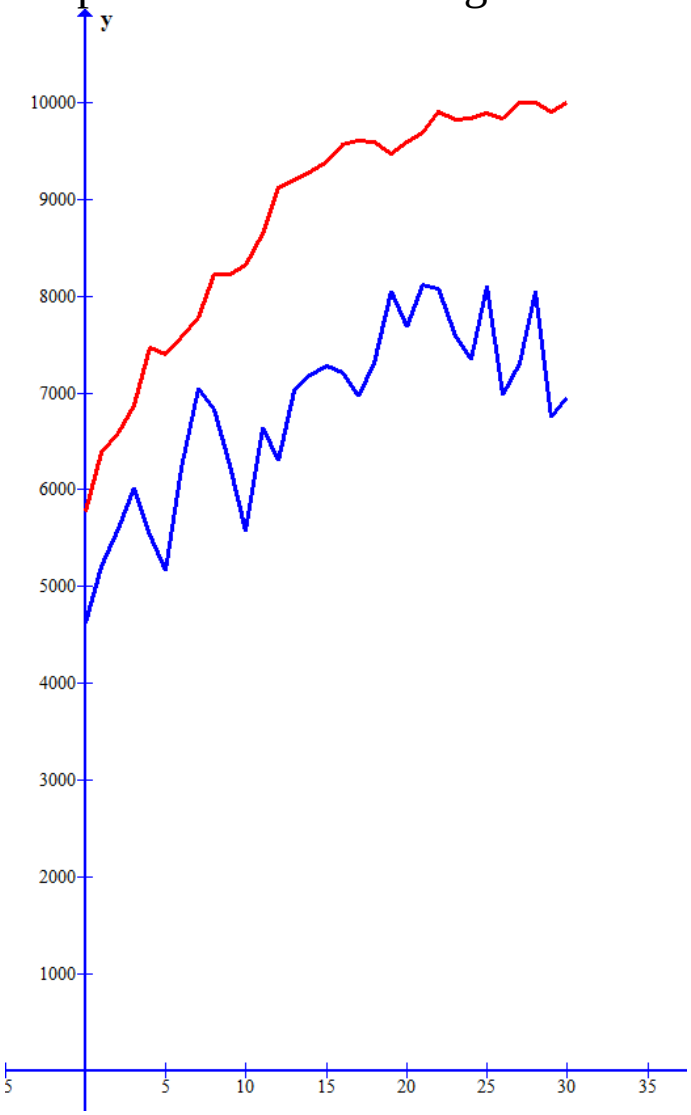
Static table of price and size:

5	3	3	8	8	3	6	5	8	2	9	9	5	7	7	10	4	8	3	8
6	2	6	9	2	4	1	4	8	4	2	10	1	1	1	7	6	1	1	4

Best chromosome from first, last and 3 intermediate generations:

First	84	91	43	18	95	22	67	44	77	9	62	30	43	1	90	95	8	65	73	91
2	73	50	72	61	94	33	31	98	90	30	92	98	47	96	29	75	99	72	5	45
3	77	92	72	76	94	71	31	98	67	88	20	98	92	100	100	75	99	72	90	94
4	77	92	72	76	94	71	92	98	67	88	65	98	92	100	100	75	99	72	90	90
Last	77	81	100	100	94	41	82	98	72	91	65	98	92	100	100	75	99	93	90	90

Graph of best and average fitness – generations:



Now let's try some experiment:

Static table of price and size:

10	10	10	10	10	10	10	10	10	10	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

(price of 1-10 items = 10, of 10-20 items =1);
(valume of all items = 1);
(size of backpack = 1200);

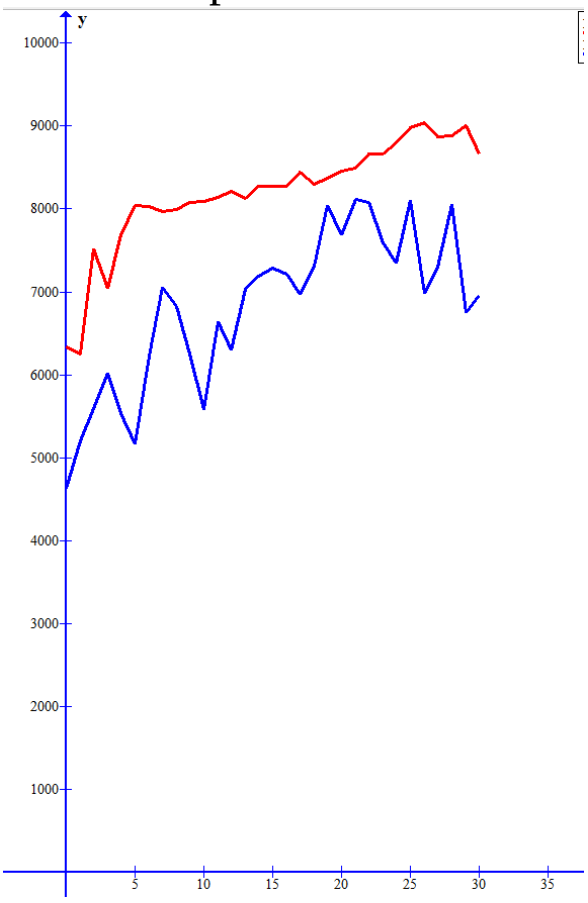
Now we can check our algorism. If an algorism is right, final chromosome will chose first half of items (because first half has better price for each item);

Lets check:

Best chromosome from first, last and 3 intermediate generations:

First	48	98	91	46	77	3	39	83	85	19	27	9	72	88	34	23	24	36	28	58
2	77	77	67	43	74	93	70	87	91	92	23	25	8	31	85	69	34	8	51	62
3	55	77	67	79	74	93	70	87	85	92	68	21	8	31	12	69	1	81	84	20
4	95	22	67	79	74	89	70	87	85	92	68	19	8	31	12	69	6	81	84	25
Last	55	51	67	79	74	89	99	87	85	92	68	88	6	31	16	24	21	81	38	25

Graph of best and average fitness – generations:



So, we can see that our algorism has chosen the first half of items.

Algorism is correct.