Діофантові рівняння — це поліноміальні рівняння з цілими коефіцієнтами в яких невідомі змінні можуть приймати тільки цілі значення. Загальний вигляд Діофантового рівняння такий:

$$k_1 x_1 + k_2 x_2 + ... + k_n x_n = res$$

Генетичний алгоритм — це еволюційний алгоритм пошуку, що використовується для вирішення задач оптимізації і моделювання шляхом послідовного підбору, комбінування і варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію. Особливістю генетичного алгоритму є використання операторів кросинговеру та мутації, які виконують пошук на множині розв'язків. Засновником генетичних алгоритмів вважається Джон Голланд, книга якого "Адаптація в природних і штучних системах" є фундаментальною в цій сфері досліджень.

Один із варіантів генетичного алгоритму розв'язку Діофантового рівняння виглядає наступним чином:

1. Вибір початкових параметрів:
   - n (розмір Діофантового рівняння);
   - k (масив коефіцієнтів Діофантового рівняння);
   - res (права частина Діофантового рівняння);
   - CHROMOSOMES_IN_GENERATION (кількість осіб у поколінні);
   - P_MUTATION (ймовірність мутацій);
   - maxGenerationCount (максимальна кількість ітерацій).
2. Вибір початкової популяції випадковим чином.
3. Обчислення цільової функції для кожної особи.

   $F_{цільова} = |res - \sum_{i=0}^{n} x_i * k_i|$, де x — вектор невідомих;
4. На основі цільової функції обчислення ймовірності вибору кожної особи. Чим менша цільова функція — тим більша ймовірність вибору.
5. Вибір CHROMOSOMES_IN_GENERATION пар батьків, згідно з відповідними імовірностями вибору.
6. Проведення кросинговеру в кожній парі батьків та вибір одного з кожної пари нащадків. В даній реалізації для кожної пари проводиться обмін лівими половинами рядка невідомих.
7. Мутація (деяка випадкова зміна) кожного числа кожної особи з імовірністю P_MUTATION.
8. Якщо для жодної особи цільова функція не рівна нулю, то перехід до п.3, інакше — вивід результатів.

Спробуємо розв'язати наступне Діофантове рівняння:

$$7x_1 + 15x_2 + 23x_3 = 113$$

Для цього задамо такі параметри:

- n = 3;
- k = {7, 15, 23};
- res = 113;
- CHROMOSOMES_IN_GENERATION = 10;
- P_MUTATION = 0.25;
- maxGenerationCount = 1000000;

Розв'язок такого рівняння програма знайшла за 523028 ітерацій, що видно на рис.1. Вектор знайдених невідомих:

x[1] = 1;
x[2] = 4;
x[3] = 2;

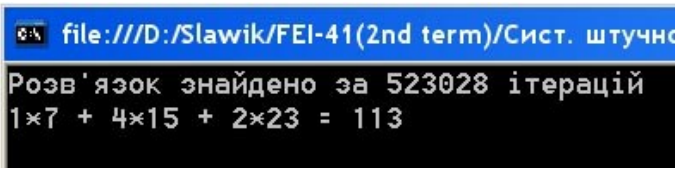

Рис. 1. Результат розв'язання Діофантового рівняння генетичним алгоритмом

Текст програми пошуку розв'язку Діофантового рівняння генетичним алгоритмом, написаний в оболонці Visual Studio 2008 мовою програмування C#:

Program.cs:

```
using System;

namespace Diofant
{
    class Program
    {
        static void Main(string[] args)
        {
            int n = 3;
            int[] koef = new int[] { 7,15,23 };
            int res = 113;

            GA ga = new GA(n, koef, res);
            int genCount = 0;
            int[] result = ga.Solve(1000000, ref genCount);

            if (result != null)
            {
                Console.WriteLine("Розв'язок знайдено за {0} ітерацій",
genCount);

                Console.Write("{0}*{1}", result[0], koef[0]);
                for (int i = 1; i < n; i++)
                {
                    Console.Write(" + {0}*{1}", result[i], koef[i]);
                }
                Console.WriteLine(" = {0}", res);
            }
            else
            {
                Console.WriteLine("Розв'язку не знайдено");
            }

            Console.ReadKey();
        }
    }
}
```

GA.cs:

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

namespace Diofant
{
    public class GA
    {
        private Random r;
        private const int CHROMOSOMES_IN_GENERATION = 10;
        private const double P_MUTATION = 0.25;
        private int n;
        private int[] koef;
        private int res;

        public GA(int n, int[] koef, int res)
        {
            this.n = n;
            this.koef = koef;
            this.res = res;
            this.r = new Random((int)DateTime.Now.Ticks);
        }

        private int F(int[] a)
        {
            int sum = 0;
            for (int i = 0; i < n; i++)
            {
                sum += a[i] * koef[i];
            }
            return Math.Abs(res - sum);
        }

        private void GetFirstGeneration(ref List<int[]> generation)
        {
            for (int i = 0; i < CHROMOSOMES_IN_GENERATION; i++)
            {
                int[] chromosome = new int[n];
                for (int j = 0; j < n; j++)
                {
                    chromosome[j] = r.Next(res) + 1;
                }
                generation.Add(chromosome);
            }
        }

        private void Crossingover(ref List<int[]> generation)
        {
            int i;

            // Рахуємо пристосованість
            double[] fitness = new double[CHROMOSOMES_IN_GENERATION];
            for (i = 0; i < CHROMOSOMES_IN_GENERATION; i++)
            {
                fitness[i] = F(generation[i]);
            }

            double sum = 0.0;
            for (i = 0; i < CHROMOSOMES_IN_GENERATION; i++)
            {
                fitness[i] = 1.0 / fitness[i];
                sum += fitness[i];
            }

            for (i = 0; i < CHROMOSOMES_IN_GENERATION; i++)
```

```
    {
        fitness[i] /= sum;
    }

    // Вибираємо батьків
    List<int> mother = new List<int>();
    List<int> father = new List<int>();

    i = 0;
    while (mother.Count != CHROMOSOMES_IN_GENERATION)
    {
        if (r.NextDouble() <= fitness[i]) mother.Add(i);
        if (i != CHROMOSOMES_IN_GENERATION - 1) i++; else i = 0;
    }

    i = 0;
    while (father.Count != CHROMOSOMES_IN_GENERATION)
    {
        if (r.NextDouble() <= fitness[i]) father.Add(i);
        if (i != CHROMOSOMES_IN_GENERATION - 1) i++; else i = 0;
    }

    // Проводимо схрещування
    List<int[]> newGeneration = new List<int[]>();
    for (int j = 0; j < CHROMOSOMES_IN_GENERATION; j++)
    {
        int[] c1 = generation[mother[j]];
        int[] c2 = generation[father[j]];
        int temp;

        for (i = 0; i < n / 2; i++)
        {
            temp = c1[i];
            c1[i] = c2[i];
            c2[i] = temp;
        }

        if (r.NextDouble() <= 0.5)
            newGeneration.Add(c1);
        else
            newGeneration.Add(c2);
    }

    generation = newGeneration;
}

private void Mutation(ref List<int[]> generation)
{
    for (int i = 0; i < CHROMOSOMES_IN_GENERATION; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (r.NextDouble() <= P_MUTATION)
            {
                int randomNumber = r.Next(res) + 1 - res / 2;
                int resNumber = generation[i][j] + randomNumber;
                if (resNumber <= 0) resNumber = 1;
                if (resNumber > res) resNumber = res;
                generation[i][j] = resNumber;
            }
        }
    }
}

public int[] Solve(int maxGenerationCount, ref int genCount)
{
```

```
            List<int[]> generation = new List<int[]>();
            GetFirstGeneration(ref generation);
            for (int i = 0; i < maxGenerationCount; i++)
            {
                Crossingover(ref generation);
                Mutation(ref generation);

                for (int j = 0; j < CHROMOSOMES_IN_GENERATION; j++)
                {
                    if (F(generation[j]) == 0)
                    {
                        genCount = i;
                        return generation[j];
                    }
                }
            }
            return null;
        }
    }
}
```