C# Y .NET 10
Parte 10. Algoritmos Evolutivos

2025-12

Rafael Alberto Moreno Parra
ramsoftware@gmail.com

Contenido

Tabla de ilustraciones

Acerca del autor

Rafael Alberto Moreno Parra

ramsoftware@gmail.com o enginelife@hotmail.com

Sitio Web: http://darwin.50webs.com (dedicado a la investigación de algoritmos evolutivos y vida artificial).

Github: https://github.com/ramsoftware

Youtube: https://www.youtube.com/@RafaelMorenoP

Licencia de este libro

Licencia del software

Todo el software desarrollado aquí tiene licencia LGPL "Lesser General Public License" [1]

Marcas registradas

En este libro se hace uso de las siguientes tecnologías registradas:

Microsoft ® Windows ® Enlace: http://windows.microsoft.com/en-US/windows/home

Microsoft ® Visual Studio 2022 ® Enlace: https://visualstudio.microsoft.com/es/vs/

Dedicatoria

A mis padres, a mi hermana….

Y a mi tropa gatuna: Suini, Grisú, Milú, Arián, Frac y mis recordados Sally, Capuchina, Tinita, Tammy, Vikingo y Michu.

Definición

Los algoritmos evolutivos, inspirados en la biología, sirven para resolver problemas que pueden llegar a ser bastantes complejos. Su mecánica es la aplicación de principios evolutivos como la reproducción y la selección natural de soluciones a un problema. A medida que pasa el tiempo (en algoritmia es en cada iteración o ciclo), se van encontrando mejores soluciones a ese problema dado. La utilidad de este tipo de algoritmos es que ofrecen un camino a encontrar soluciones mejores (no las ideales) a problemas complejos donde una solución clásica o sistemática tardaría muchísimo. En la parte de “Simulaciones” se trabajó con soluciones de tipo Montecarlo para hallar la ruta de menor costo, o resolver un Sudoku, o hallar el área bajo la curva. Los algoritmos evolutivos siguen ese estilo.

Búsqueda aleatoria

Está la siguiente cadena:

esta es una prueba de texto

¿Es posible generar cadenas de caracteres al azar de tal manera que coincida con la cadena anterior? En realidad, sí, pero es altamente improbable que la cadena generada al azar acierte a esa cadena en particular.

Probando con código:

J/001.cs

```
namespace Ejemplo
{
 internal class Program
 {
  static void Main(string[] args)
  {
   /* Único generador de números aleatorios para todo el programa */
   Random Azar = new();

   /* Cadena original a "adivinar" */
   string Original = "naturaleza";

   /* Iniciar el proceso */
   Proceso(Azar, Original);
  }

  /* Método que realiza el proceso de "adivinar" la cadena original */
  static void Proceso(Random Azar, string Original) {
   int Contador = 0;
   for (; ;) {

    /* Generar una cadena aleatoria */
    string Cadena = GenerarCadenaAzar(Azar, Original.Length);

    /* Comparar con la original */
    if (string.Compare(Original, Cadena) == 0) {
     break;
    }

    /* Incrementar el contador e informar cada 1000 intentos */
    Contador++;
    if (Contador % 1000 == 0) {
     Console.WriteLine($"Intentos: {Contador:N0}");
    }
   }
```


Autor: Rafael Alberto Moreno Parra

```
    }

    /* Método que genera una cadena aleatoria de una longitud dada */
    static string GenerarCadenaAzar(Random Azar, int longitud) {
      char[] caracteres = new char[longitud];
      for (int i = 0; i < longitud; i++) {
        caracteres[i] = LetraAzar(Azar);
      }
      return new string(caracteres);
    }

    /* Método que genera una letra minúscula aleatoria */
    static char LetraAzar(Random Azar) {
      return (char)Azar.Next('a', 'z' + 1);
    }
  }
}
```

Es muy improbable que la cadena generada al azar acierte a la cadena original, luego es muy improbable el programa se detenga.

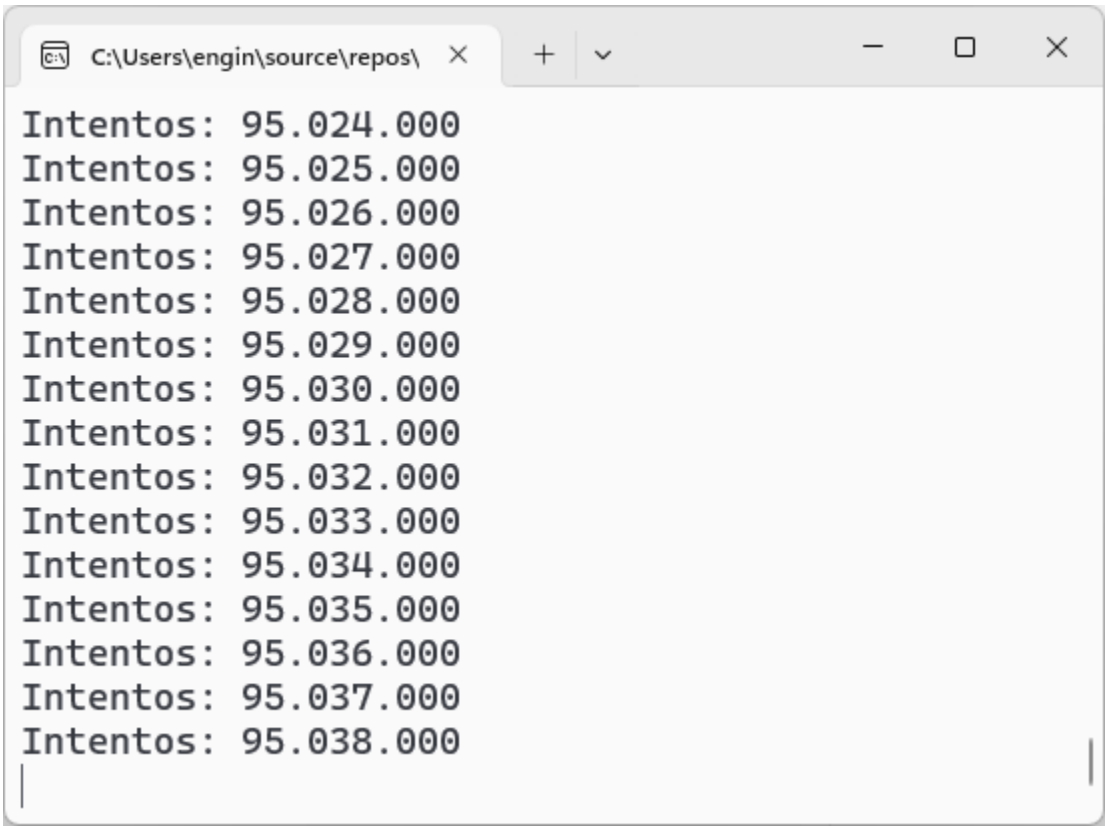

Ilustración 1: El programa es muy improbable que se detenga

El operador Mutación

Como el algoritmo anterior falla en dar con la cadena. Se hace uso entonces del concepto "algoritmo evolutivo", en este caso, se aplica el operador mutación.

El algoritmo:

```
Algoritmo Evolutivo
Inicio
 Generar población de N individuos al azar
 Inicio ciclo
  Seleccionar al azar dos individuos de esa población: A y B
  Evaluar adaptación de A
  Evaluar adaptación de B
  Si adaptación de A es mejor que adaptación de B entonces
   Eliminar individuo B
   Duplicar individuo A
   Modificar levemente al azar el nuevo duplicado
  de lo contrario
   Eliminar individuo A
   Duplicar individuo B
   Modificar levemente al azar el nuevo duplicado
  Fin Si
 Fin ciclo
 Buscar individuo con mejor adaptación de la población
 Imprimir individuo
Fin
```

Explicación del algoritmo:

"Generar población de N individuos al azar": Generar N cadenas al azar y guardarlas en un arreglo o lista.

"Evaluar adaptación de": Evaluar cuantitativamente esa cadena con respecto a la original. En ese caso, una medida puede ser sumar un punto por cada letra que coincida con la cadena original. A mayor puntaje, mejor es la adaptación.

Al principio se crea una población con varios individuos generados al azar, por ejemplo, con 10 individuos:

```
1: oTGBVáDgzD!óxëVoÓsbYAoVVRwkTj
2: Áh?JÄgÉmmÍ¡!óQUbFtdfymNtNSjmF
3: rEVáa Z¡oaÓC¿rxy?Äx¿pÑjíHmfGÓ
4: ÜkJñzU¡mCoQEZe! ÄmúÖN¿Ghj zoá
5: tWsLótMRFhoB¡íÄpkkDgYq¡LrXëÓU
6: LFopAÏéÓüYj!ëÖÄvÚMycCÄÍÏFPLÜ?
```


Autor: Rafael Alberto Moreno Parra

```
 7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsíb
 8: jOYnÚtAEH¿Ë¡HVvPB ÁëÜáx¿fuÜch
 9: ñróQEx¡MqöúïEKÜïNTpRBóBWúofsö
10: ëWÄUPLúíïBVëÁÑLaómpKPÓ!ÏWXYÚQ
```

Se seleccionan dos individuos al azar, por ejemplo, el 3 y el 9:

```
 1: oTGBVáDgzD!óxëVoÓsbYAoVVRwkTj
 2: Áh?JÄgÉmmÍ¡!óQUbFtdfymNtNSjmF
 3: rEVáa Z¡oaÓC¿rxy?Äx¿pÑjíHmfGÓ
 4: ÜkJñzU¡mCoQEZe! ÄmúÖN¿Ghj zoá
 5: tWsLótMRFhoB¡íÄpkkDgYq¡LrXëÓU
 6: LFopAÏéÓüYj!ëÖÄvÚMycCÄÍÏFPLÜ?
 7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsíb
 8: jOYnÚtAEH¿Ë¡HVvPB ÁëÜáx¿fuÜch
 9: ñróQEx¡MqöúïEKÜïNTpRBóBWúofsö
10: ëWÄUPLúíïBVëÁÑLaómpKPÓ!ÏWXYÚQ
```

Ahora se evalúa cual individuo, de los dos seleccionados, es el mejor. Una forma de hacerlo es comparar letra por letra entre la cadena del individuo y la cadena original, si coinciden entonces suma 1 al puntaje.

Evaluar: `rEVáa Z¡oaÓC¿rxy?Äx¿pÑjíHmfGÓ` vs `Esta es una prueba de texto` Es 1

Evaluar: `ñróQEx¡MqöúïEKÜïNTpRBóBWúofsö` vs `Esta es una prueba de texto` Es 0

El individuo ganador es `rEVáa Z¡oaÓC¿rxy?Äx¿pÑjíHmfGÓ` con puntaje de 1, el otro pierde con un puntaje de 0

Autor: Rafael Alberto Moreno Parra

El individuo perdedor es eliminado de la población

1: oTGBVáDgzD!óxëVoÓsbYAoVVRwkTj

2: Áh?JÄgÉmmÍ¡!óQUbFtdfymNtNSjmF

3: rEVáa Z¡oaÓC¿rxy?Äx¿pÑjíHmfGÓ

4: ÜkJñzU¡mCoQEZe! ÄmúÖN¿Ghj zoá

5: tWsLótMRFhoB¡íÄpkkDgYq¡LrXëÓU

6: LFopAÏéÓüYj!ëÖÄvÚMycCÄÍÏFPLÜ?

7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsíb

8: jOYnÚtAEH¿Ë¡HVvPB ÁëÜáx¿fuÜch

~~9: ñróQEx¡MqöúïEKÜïNTpRBóBWúofsö~~

10: ëWÄUPLúíïBVëÁÑLaómpKPÓ!ÏWXYÚQ

El mejor individuo se premia creando una copia de sí mismo

3: rEVáa Z¡oaÓC¿rxy?Äx¿pÑjíHmfGÓ

Ahora esa copia se modifica en algún punto al azar

¿?: rEVáa Z**h**oaÓC¿rxy?Äx¿pÑjíHmfGÓ

Esa copia modificada ahora hace parte de la población ocupando el espacio dejado por el perdedor

1: oTGBVáDgzD!óxëVoÓsbYAoVVRwkTj

2: Áh?JÄgÉmmÍ¡!óQUbFtdfymNtNSjmF

3: rEVáa Z¡oaÓC¿rxy?Äx¿pÑjíHmfGÓ

4: ÜkJñzU¡mCoQEZe! ÄmúÖN¿Ghj zoá

5: tWsLótMRFhoB¡íÄpkkDgYq¡LrXëÓU

6: LFopAÏéÓüYj!ëÖÄvÚMycCÄÍÏFPLÜ?

7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsíb

8: jOYnÚtAEH¿Ë¡HVvPB ÁëÜáx¿fuÜch

9: rEVáa ZhoaÓC¿rxy?Äx¿pÑjíHmfGÓ

10: ëWÄUPLúíïBVëÁÑLaómpKPÓ!ÏWXYÚQ

El proceso se repite varias veces hasta que se encuentra el mejor individuo que soluciona el problema.

```
namespace Ejemplo {
 internal class Program {
  public static char[] Letras;

  static void Main() {
   /* Único generador de números aleatorios para todo el programa */
   Random Azar = new();

   /* Alfabeto de letras */
   Letras = "abcdefghijklmnopqrstuvwxyz ".ToCharArray();

   /* Cadena original a "adivinar" */
   string Original = "esta es una prueba para dar con una cadena";

   /* Iniciar el proceso */
   Proceso(Azar, Original);
  }

  /* Método que realiza el proceso de "adivinar" la cadena original */
  static void Proceso(Random Azar, string Original) {
   /* Trabaja con arreglos de caracteres */
   char[] OriginalArray = Original.ToCharArray();

   /* Genera población inicial con individuos que tengan la
      misma longitud de la cadena original */
   int TotalIndividuos = 1000;
   char[][] Individuos = new char[TotalIndividuos][];

   int Tamano = OriginalArray.Length;
   for (int indiv = 0; indiv < TotalIndividuos; indiv++) {
    Individuos[indiv] = new char[Tamano];
```


Autor: Rafael Alberto Moreno Parra

```
    for (int pos = 0; pos < Tamano; pos++)
      Individuos[indiv][pos] = Letras[Azar.Next(Letras.Length)];
  }

  int Contador = 0;
  int MejorPuntaje = -1;
  int MejorIndividuo = -1;

  while (true) {

    /* Seleccionar dos inviduos aleatoriamente */
    int Indice1 = Azar.Next(TotalIndividuos);
    int Indice2;
    do {
      Indice2 = Azar.Next(TotalIndividuos);
    } while (Indice2 == Indice1);

    /* Evalúa cada individuo */
    int Puntaje1 = Puntuar(Individuos[Indice1], OriginalArray);
    int Puntaje2 = Puntuar(Individuos[Indice2], OriginalArray);

    if (Puntaje1 > MejorPuntaje) { MejorPuntaje = Puntaje1; MejorIndividuo = Indice1; }
    if (Puntaje2 > MejorPuntaje) { MejorPuntaje = Puntaje2; MejorIndividuo = Indice2; }

    /* Evalúa si un individuo ha acertado a la cadena original */
    if (Puntaje1 == OriginalArray.Length || Puntaje2 == OriginalArray.Length) {
      break;
    } /* Evalúa si el individuo 1 es mejor que el individuo 2 */
    else if (Puntaje1 > Puntaje2) {
      // El ganador sobre-escribe al perdedor
      Array.Copy(Individuos[Indice1], Individuos[Indice2], Individuos[Indice1].Length);

      //Modifica la copia
      Individuos[Indice1][Azar.Next(OriginalArray.Length)] = Letras[Azar.Next(Letras.Length)];
    } /* Evalúa si el individuo 2 es mejor que el individuo 1 */
    else if (Puntaje2 > Puntaje1) {
```

```
      // El ganador sobre-escribe al perdedor
      Array.Copy(Individuos[Indice2], Individuos[Indice1], Individuos[Indice2].Length);

      //Modifica la copia
      Individuos[Indice2][Azar.Next(OriginalArray.Length)] = Letras[Azar.Next(Letras.Length)];
     }

     /* Incrementar el contador e informar cada 1000 intentos */
     Contador++;
     if (Contador % 1000 == 0) {
      Console.WriteLine($"Intento: {Contador:N0} Mejor individuo: [{new string(Individuos[MejorIndividuo])}]");
     }
    }

    Console.WriteLine($"Intento: {Contador:N0} Mejor individuo: [{new string(Individuos[MejorIndividuo])}]");
   }

   /* Evalúa la similitud entre dos arreglos de caracteres */
   static int Puntuar(char[] charArreglo1, char[] charArreglo2) {
    int Similitud = 0;
    for (int i = 0; i < charArreglo1.Length; i++) {
     if (charArreglo1[i] == charArreglo2[i]) {
      Similitud++;
     }
    }
    return Similitud;
   }
  }
}
```


Autor: Rafael Alberto Moreno Parra

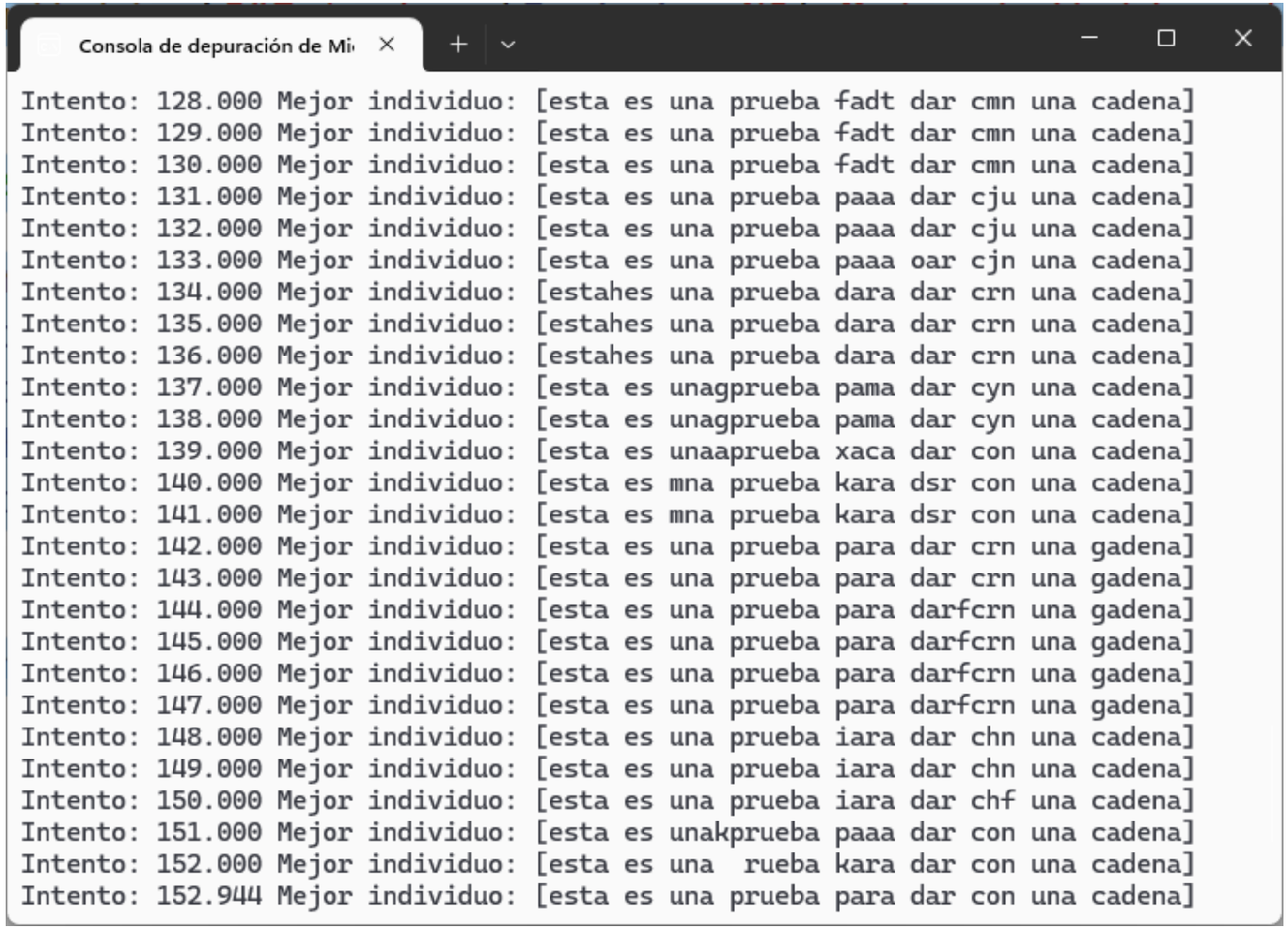

Ilustración 2: Algoritmo evolutivo con el operador mutación

Después de varios ciclos, se logró dar con la cadena.

El operador Cruce

Dada la cadena de destino:

```
String original = "¡Esta es una prueba de texto!";
```

Se genera una población donde cada individuo es una cadena, por ejemplo:

```
 1: oTGBVáDgzD!óxëVoÓsbYAoVVRwkTj
 2: Áh?JÄgÉmmÍ¡!óQUbFtdfymNtNSjmF
 3: rEVáaqZ¡oaÓC¿rxy?Äx¿pÑjíHmfGÓ
 4: ÜkJñzU¡mCoQEZe! ÄmúÖN¿Ghj zoá
 5: tWsLótMRFhoB¡íÄpkkDgYq¡LrXëÓU
 6: LFopAÏéÓüYj!ëÖÄvÚMycCÄÍÏFPLÜ?
 7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsíb
 8: jOYnÚtAEH¿Ë¡HVvPB ÁëÜáx¿fuÜch
 9: ñróQEx¡MqöúïEKÜïNTpRBóBWúofsö
10: ëWÄUPLúíïBVëÁÑLaómpKPÓ!ÏWXYÚQ
```

Se seleccionan dos individuos al azar

```
 1: oTGBVáDgzD!óxëVoÓsbYAoVVRwkTj
 2: Áh?JÄgÉmmÍ¡!óQUbFtdfymNtNSjmF
 3: rEVáaqZ¡oaÓC¿rxy?Äx¿pÑjíHmfGÓ
 4: ÜkJñzU¡mCoQEZe! ÄmúÖN¿Ghj zoá
 5: tWsLótMRFhoB¡íÄpkkDgYq¡LrXëÓU
 6: LFopAÏéÓüYj!ëÖÄvÚMycCÄÍÏFPLÜ?
 7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsíb
 8: jOYnÚtAEH¿Ë¡HVvPB ÁëÜáx¿fuÜch
 9: ñróQEx¡MqöúïEKÜïNTpRBóBWúofsö
10: ëWÄUPLúíïBVëÁÑLaómpKPÓ!ÏWXYÚQ
```


Autor: Rafael Alberto Moreno Parra

Esos dos individuos generan dos hijos con la operación de cruce:

Hijo A

Al azar se escoge un punto de corte, la primera parte la pone el individuo A y el resto el Individuo B (en fondo verde son las partes de los individuos que constituirán el Hijo A)

rEVáaqZ¡oaÓC¿rxy?Äx¿pÑjíHmfGÓ y ñróQEx¡MqöúiEKÜiNTpRBóBWúofsö

Hijo A nace al unir las dos piezas: rEVáax¡MqöúiEKÜiNTpRBóBWúofsö

Hijo B

Usando el mismo punto de corte anterior, la primera parte la pone el individuo B y el resto el Individuo A (en fondo azul son las partes de los individuos que constituirán el Hijo B)

rEVáaqZ¡oaÓC¿rxy?Äx¿pÑjíHmfGÓ y ñróQEx¡MqöúiEKÜiNTpRBóBWúofsö

Hijo B nace al unir las dos piezas: ñróQEqZ¡oaÓC¿rxy?Äx¿pÑjíHmfGÓ

Ambos hijos son evaluados, si superan a los padres en puntaje, entonces esos hijos reemplazan a sus padres.

```
namespace Ejemplo {
 internal class Program {
  public static char[] Letras;

  static void Main() {
   /* Único generador de números aleatorios para todo el programa */
   Random Azar = new();

   /* Alfabeto de letras */
   Letras = "abcdefghijklmnopqrstuvwxyz ".ToCharArray();

   /* Cadena original a "adivinar" */
   string Original = "esta es una prueba para dar con una cadena";

   /* Iniciar el proceso */
   Proceso(Azar, Original);
  }

  /* Método que realiza el proceso de "adivinar" la cadena original */
  static void Proceso(Random Azar, string Original) {
   /* Trabaja con arreglos de caracteres */
   char[] OriginalArray = Original.ToCharArray();

   /* Genera población inicial con individuos que tengan la
      misma longitud de la cadena original */
   int TotalIndividuos = 1000;
   char[][] Individuos = new char[TotalIndividuos][];

   int Tamano = OriginalArray.Length;
   for (int indiv = 0; indiv < TotalIndividuos; indiv++) {
    Individuos[indiv] = new char[Tamano];
    for (int pos = 0; pos < Tamano; pos++)
```

```
      Individuos[indiv][pos] = Letras[Azar.Next(Letras.Length)];
    }

    int Contador = 0;
    int MejorPuntaje = -1;
    int MejorIndividuo = -1;

    while (true) {

     /* Seleccionar dos inviduos aleatoriamente */
     int Indice1 = Azar.Next(TotalIndividuos);
     int Indice2;
     do {
      Indice2 = Azar.Next(TotalIndividuos);
     } while (Indice2 == Indice1);

     /* Genera los hijos */
     char[] HijoA = new char[Tamano];
     int Pos = Azar.Next(Tamano);

     // Copiar desde Padre [0..Pos]
     Array.Copy(Individuos[Indice1], 0, HijoA, 0, Pos + 1);

     // Copiar desde Madre [Pos+1..fin]
     Array.Copy(Individuos[Indice2], Pos + 1, HijoA, Pos + 1, Individuos[Indice2].Length - (Pos + 1));

     char[] HijoB = new char[Tamano];

     // Copiar desde Padre [0..Pos]
     Array.Copy(Individuos[Indice2], 0, HijoB, 0, Pos + 1);

     // Copiar desde Madre [Pos+1..fin]
     Array.Copy(Individuos[Indice1], Pos + 1, HijoB, Pos + 1, Individuos[Indice1].Length - (Pos + 1));

     /* Evalúa cada individuo */
     int Puntaje1 = Puntuar(Individuos[Indice1], OriginalArray);
```

```
    int Puntaje2 = Puntuar(Individuos[Indice2], OriginalArray);
    int PuntajeHijoA = Puntuar(HijoA, OriginalArray);
    int PuntajeHijoB = Puntuar(HijoB, OriginalArray);

    /* Si el hijo es mejor que algún progenitor, entonces se sobre-escribe el progenitor */
    if (PuntajeHijoA > Puntaje1)
     Array.Copy(HijoA, Individuos[Indice1], Individuos[Indice1].Length);
    if (PuntajeHijoA > Puntaje2)
     Array.Copy(HijoA, Individuos[Indice2], Individuos[Indice2].Length);

    if (PuntajeHijoB > Puntaje1)
     Array.Copy(HijoB, Individuos[Indice1], Individuos[Indice1].Length);
    if (PuntajeHijoB > Puntaje2)
     Array.Copy(HijoB, Individuos[Indice2], Individuos[Indice2].Length);

    if (Puntaje1 > MejorPuntaje) { MejorPuntaje = Puntaje1; MejorIndividuo = Indice1; }
    if (Puntaje2 > MejorPuntaje) { MejorPuntaje = Puntaje2; MejorIndividuo = Indice2; }

    /* Evalúa si un individuo ha acertado a la cadena original */
    if (Puntaje1 == OriginalArray.Length || Puntaje2 == OriginalArray.Length) {
     break;
    }

    /* Incrementar el contador e informar cada 1000 intentos */
    Contador++;
    if (Contador % 1000 == 0) {
     Console.WriteLine($"Intento: {Contador:N0} Mejor individuo: [{new string(Individuos[MejorIndividuo])}]");
    }
   }

   Console.WriteLine($"Intento: {Contador:N0} Mejor individuo: [{new string(Individuos[MejorIndividuo])}]");
  }

  /* Evalúa la similitud entre dos arreglos de caracteres */
  static int Puntuar(char[] charArreglo1, char[] charArreglo2) {
   int Similitud = 0;
```

```
    for (int i = 0; i < charArreglo1.Length; i++) {
     if (charArreglo1[i] == charArreglo2[i]) {
      Similitud++;
     }
    }
    return Similitud;
   }
  }
}
```

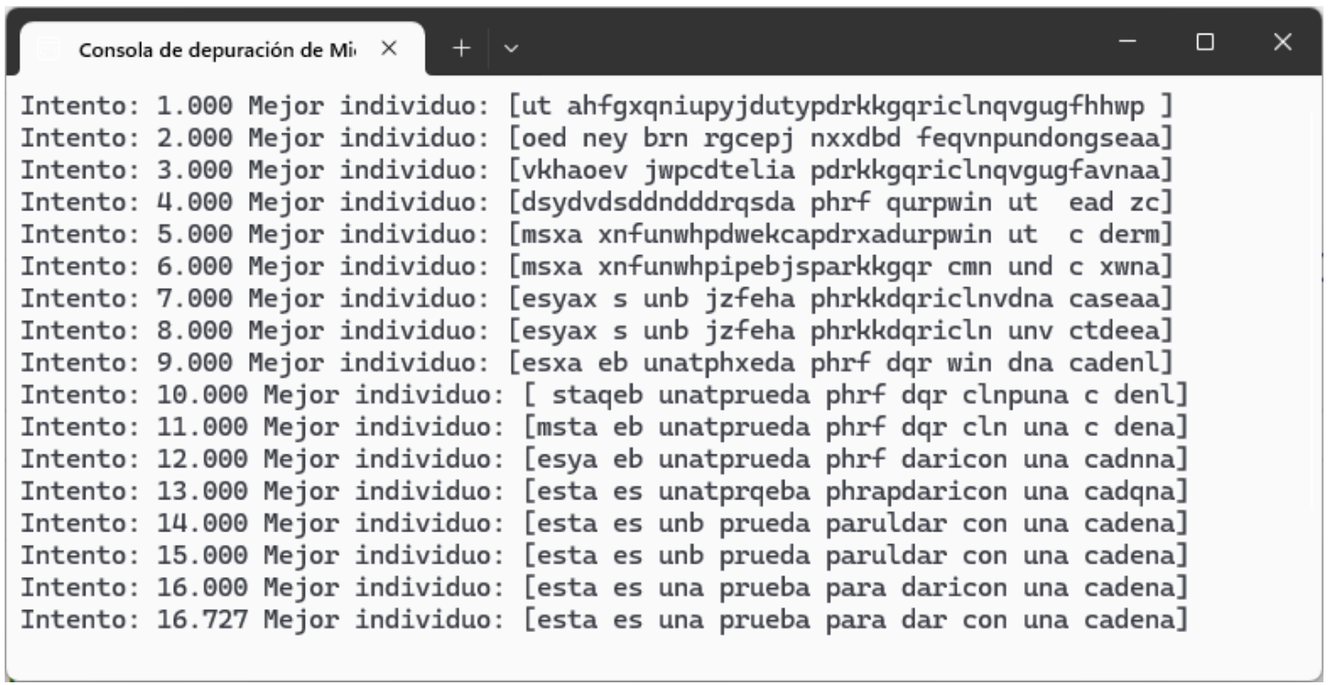

Ilustración 3: Algoritmo evolutivo con el operador cruce

¡OJO! Sucedía que, en ciertas ejecuciones, la simulación no se detenía, la razón es que si en la cadena original, si una letra en una determinada posición no aparece en esa posición en ningún individuo de la población inicial, no hay forma que los cruces logren dar con la cadena original.

Por ejemplo, obsérvese que en la posición 2 (contando desde cero) de la cadena original esta la letra "t". Puede suceder que ningún individuo de la población inicial tenga la letra "t" en la posición 2 y esto hace que no pueda generarse la cadena original.

¿Solución? Aumentar bastante la población minimiza el riesgo (pero no desaparece), sin embargo, el precio a pagar es que se tarda más en dar con la cadena original.

Combinando los operadores cruce y mutación

Mutar es confiable pero lento, cruzar es rápido, pero hay riesgo de no encontrar la cadena. La solución es combinar ambos operadores.

```
namespace Ejemplo {
 internal class Program {
  public static char[] Letras;

  static void Main() {
   /* Único generador de números aleatorios para todo el programa */
   Random Azar = new();

   /* Alfabeto de letras */
   Letras = "abcdefghijklmnopqrstuvwxyz ".ToCharArray();

   /* Cadena original a "adivinar" */
   string Original = "esta es una prueba para dar con una cadena";

   /* Iniciar el proceso */
   Proceso(Azar, Original);
  }

  /* Método que realiza el proceso de "adivinar" la cadena original */
  static void Proceso(Random Azar, string Original) {
   /* Trabaja con arreglos de caracteres */
   char[] OriginalArray = Original.ToCharArray();

   /* Genera población inicial con individuos que tengan la
      misma longitud de la cadena original */
   int TotalIndividuos = 10;
   char[][] Individuos = new char[TotalIndividuos][];

   int Tamano = OriginalArray.Length;
   for (int indiv = 0; indiv < TotalIndividuos; indiv++) {
    Individuos[indiv] = new char[Tamano];
    for (int pos = 0; pos < Tamano; pos++)
```

```
      Individuos[indiv][pos] = Letras[Azar.Next(Letras.Length)];
    }

    int Contador = 0;
    int MejorPuntaje = -1;
    int MejorIndividuo = -1;

    while (true) {

     /* Seleccionar dos inviduos aleatoriamente */
     int Indice1 = Azar.Next(TotalIndividuos);
     int Indice2;
     do {
      Indice2 = Azar.Next(TotalIndividuos);
     } while (Indice2 == Indice1);

     /* Genera los hijos */
     char[] HijoA = new char[Tamano];
     int Pos = Azar.Next(Tamano);

     // Copiar desde Padre [0..Pos]
     Array.Copy(Individuos[Indice1], 0, HijoA, 0, Pos + 1);

     // Copiar desde Madre [Pos+1..fin]
     Array.Copy(Individuos[Indice2], Pos + 1, HijoA, Pos + 1, Individuos[Indice2].Length - (Pos + 1));

     char[] HijoB = new char[Tamano];

     // Copiar desde Padre [0..Pos]
     Array.Copy(Individuos[Indice2], 0, HijoB, 0, Pos + 1);

     // Copiar desde Madre [Pos+1..fin]
     Array.Copy(Individuos[Indice1], Pos + 1, HijoB, Pos + 1, Individuos[Indice1].Length - (Pos + 1));

     /* Muta los hijos */
     HijoA[Azar.Next(OriginalArray.Length)] = Letras[Azar.Next(Letras.Length)];
```

```
    HijoB[Azar.Next(OriginalArray.Length)] = Letras[Azar.Next(Letras.Length)];

    /* Evalúa cada individuo */
    int Puntaje1 = Puntuar(Individuos[Indice1], OriginalArray);
    int Puntaje2 = Puntuar(Individuos[Indice2], OriginalArray);
    int PuntajeHijoA = Puntuar(HijoA, OriginalArray);
    int PuntajeHijoB = Puntuar(HijoB, OriginalArray);

    /* Si el hijo es mejor que algún progenitor, entonces se sobre-escribe el progenitor */
    if (PuntajeHijoA > Puntaje1)
     Array.Copy(HijoA, Individuos[Indice1], Individuos[Indice1].Length);
    if (PuntajeHijoA > Puntaje2)
     Array.Copy(HijoA, Individuos[Indice2], Individuos[Indice2].Length);

    if (PuntajeHijoB > Puntaje1)
     Array.Copy(HijoB, Individuos[Indice1], Individuos[Indice1].Length);
    if (PuntajeHijoB > Puntaje2)
     Array.Copy(HijoB, Individuos[Indice2], Individuos[Indice2].Length);

    if (Puntaje1 > MejorPuntaje) { MejorPuntaje = Puntaje1; MejorIndividuo = Indice1; }
    if (Puntaje2 > MejorPuntaje) { MejorPuntaje = Puntaje2; MejorIndividuo = Indice2; }

    /* Evalúa si un individuo ha acertado a la cadena original */
    if (Puntaje1 == OriginalArray.Length || Puntaje2 == OriginalArray.Length) {
     break;
    }

    /* Incrementar el contador e informar cada 1000 intentos */
    Contador++;
    if (Contador % 1000 == 0) {
     Console.WriteLine($"Intento: {Contador:N0} Mejor individuo: [{new string(Individuos[MejorIndividuo])}]");
    }
   }

   Console.WriteLine($"Intento: {Contador:N0} Mejor individuo: [{new string(Individuos[MejorIndividuo])}]");
  }
```

```
  /* Evalúa la similitud entre dos arreglos de caracteres */
  static int Puntuar(char[] charArreglo1, char[] charArreglo2) {
   int Similitud = 0;
   for (int i = 0; i < charArreglo1.Length; i++) {
    if (charArreglo1[i] == charArreglo2[i]) {
     Similitud++;
    }
   }
   return Similitud;
  }
 }
}
```

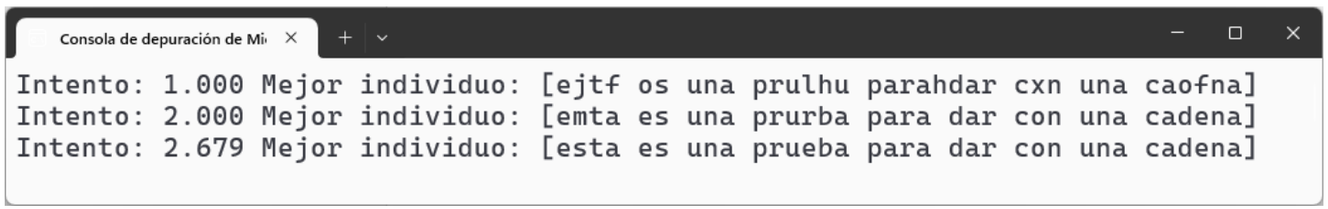

Ilustración 4: Algoritmo evolutivo: Combinando operador cruce y mutación

No solo es mucho más rápido encontrando la cadena, es que además es confiable. El ejemplo usó una población muy pequeña inicial, de diez individuos.

Buscar el máximo en una función

En este segundo problema, dada una función **y=f(x)**, un valor inicial para **x=a** y un valor final para **x=b** donde **a < b**, se pide encontrar en cuál valor de **"x"**, se obtiene el máximo valor de **"y"** en ese rango **[a, b].** El procedimiento es derivar la función **y=f(x)**, igualar a cero y así se obtiene el valor de **x**. Pero puede que **y=f(x)** sea una ecuación bastante compleja, que complique el derivarla y resolverla a **y'=0**. En estos casos es posible aplicar los algoritmos evolutivos.

Por ejemplo, se tiene esta curva:

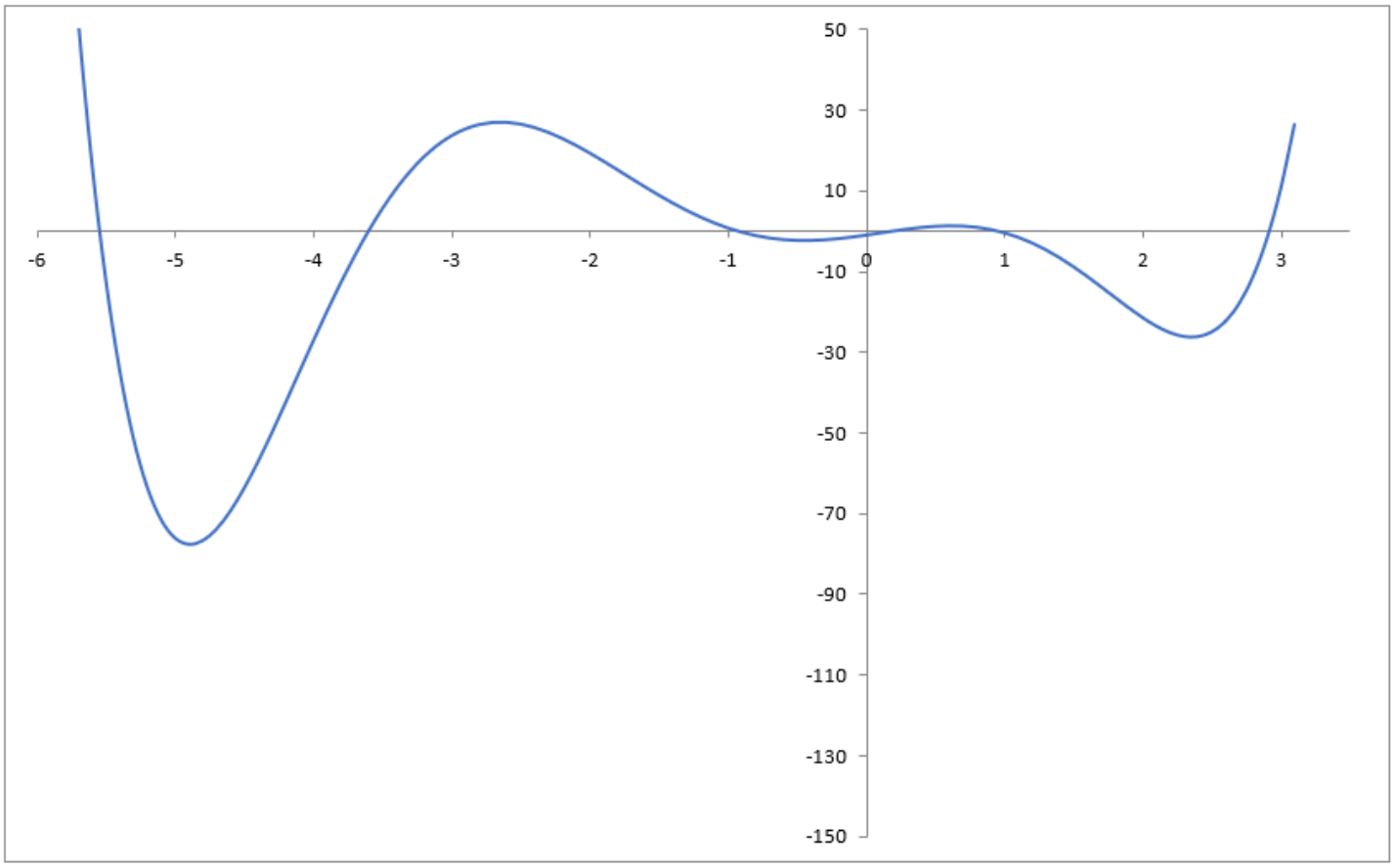

Ilustración 5:Polinomio de grado 6

Para hallar el valor de X que obtenga el mayor valor de Y en el rango Xmin y Xmax. La curva es:

$$Y = 0.1 * X^6 + 0.6 * X^5 - 0.9 * X^4 - 6.2 * X^3 + 2 * X^2 + 5 * X - 1$$

El primer paso con los algoritmos evolutivos es determinar cómo construir al individuo que represente una solución. Una forma de hacerlo es que los individuos sean números reales de doble precisión. Así que se crea una población de varios individuos como números reales aleatorios entre 0 y -1. Aplicando la distribución uniforme entonces esos valores quedarían entre **a** y **b**.

En cada ciclo del proceso, se escogen dos individuos al azar. Se aplica el operador cruce (que sería el promedio de los dos escogidos), eso genera un hijo, luego a ese hijo se le aplica el operador mutación (un ligero cambio aleatorio). Si el hijo es mejor (genera mayor valor en Y) que sus progenitores, entonces ese hijo sobre escribe a los padres.

El código es el siguiente:

```
namespace Ejemplo {
 internal class Program {

  static void Main() {
   /* Único generador de números aleatorios para todo el programa */
   Random Azar = new();

   /* Iniciar el proceso */
   double Xini = -4;
   double Xfin = 1;
   int TotalIndividuos = 100;
   int NumeroCiclos = 10000;
   MaximoValor(Azar, Xini, Xfin, TotalIndividuos, NumeroCiclos);
  }

  /* Método que realiza el proceso de encontrar el máximo valor */
  static void MaximoValor(Random Azar, double Xini, double Xfin, int TotalIndividuos, int NumeroCiclos) {

   /* Genera población inicial con individuos con valores aleatorios */
   double[] Individuos = new double[TotalIndividuos];

   for (int indiv = 0; indiv < TotalIndividuos; indiv++) {
    Individuos[indiv] = Azar.NextDouble();
   }

   int Contador = 0;
   double MejorPuntaje = double.MinValue;
   int MejorIndividuo = -1;
   double MejorValorX, MayorValorY;

   for (int ciclos = 1; ciclos <= NumeroCiclos; ciclos++) {
```

```
    /* Seleccionar dos inviduos aleatoriamente */
    int Indice1 = Azar.Next(TotalIndividuos);
    int Indice2;
    do {
     Indice2 = Azar.Next(TotalIndividuos);
    } while (Indice2 == Indice1);

    /* Evalúa cada individuo */
    double Puntaje1 = Ecuacion(Individuos[Indice1] * (Xfin - Xini) + Xini);
    double Puntaje2 = Ecuacion(Individuos[Indice2] * (Xfin - Xini) + Xini);

    if (Puntaje1 > MejorPuntaje) { MejorPuntaje = Puntaje1; MejorIndividuo = Indice1; }
    if (Puntaje2 > MejorPuntaje) { MejorPuntaje = Puntaje2; MejorIndividuo = Indice2; }

    /* Genera un hijo */
    double Hijo = (Individuos[Indice1] + Individuos[Indice2]) / 2.0;

    /* Muta el hijo */
    Hijo += (Azar.NextDouble() - 0.5) * 0.1;

    /* Evalúa el hijo */
    double PuntajeHijo = Ecuacion(Hijo * (Xfin - Xini) + Xini);

    /* Si el hijo es mejor que algún progenitor, entonces se sobre-escribe el progenitor */
    if (PuntajeHijo > Puntaje1)
     Individuos[Indice1] = Hijo;

    if (PuntajeHijo > Puntaje2)
     Individuos[Indice2] = Hijo;

    /* Incrementar el contador e informar cada 1000 intentos */
    Contador++;
    if (Contador % 1000 == 0) {
     MejorValorX = Individuos[MejorIndividuo] * (Xfin - Xini) + Xini;
     MayorValorY = Ecuacion(MejorValorX);
     Console.WriteLine($"Intento: {Contador:N0} Mejor individuo: [{MejorValorX}] con Valor: [{MayorValorY}]");
```

```
      }
    }

    MejorValorX = Individuos[MejorIndividuo] * (Xfin - Xini) + Xini;
    MayorValorY = Ecuacion(MejorValorX);
    Console.WriteLine($"Intento: {Contador:N0} Mejor individuo: [{MejorValorX}] con Valor: [{MayorValorY}]");
   }

   static double Ecuacion(double x) {
    return 0.1 * Math.Pow(x, 6) + 0.6 * Math.Pow(x, 5) + (-0.9 * Math.Pow(x, 4)) - 6.2 * Math.Pow(x, 3) + 2 * x * x + 5 * x - 1;
   }
  }
}
```

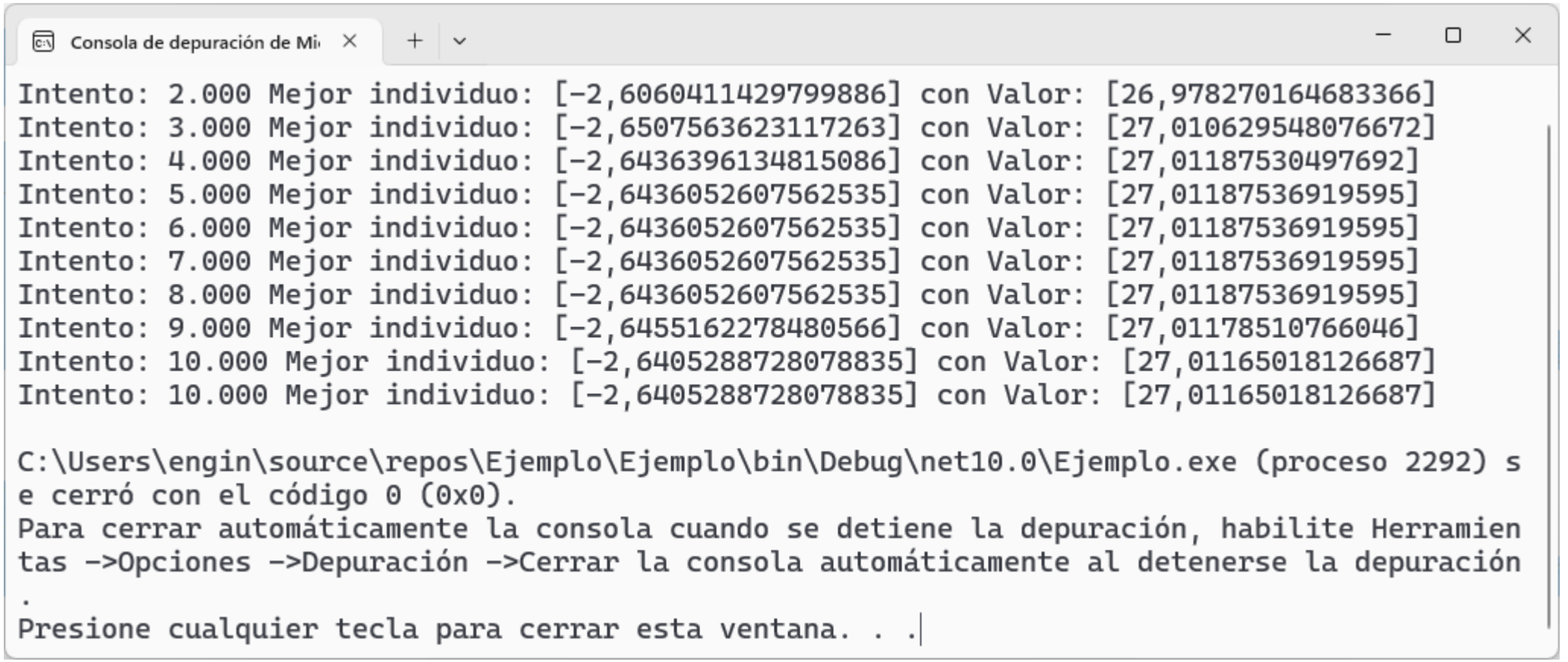

Ilustración 6: Buscar el máximo en una función

De genotipos y fenotipos

En el anterior problema, los individuos fueron números reales de doble precisión en C#, la mutación fue adicionar o sustraer un valor aleatorio. ¿Y si se quiere mayor control en esa modificación para poder aplicar el operador cruce?

En los seres vivos, las instrucciones para su desarrollo y funcionamiento están en el ADN. Esa información genética es el genotipo. La interpretación de ese genotipo es el fenotipo. Un ejemplo grosso modo:

Genotipo	**Fenotipo**
ACCTGAACTTGGCCAATGGCCTAACCTG	Forma del pico de un ave

En algoritmos evolutivos se hace una analogía similar: el genotipo es una cadena de números binarios, esta es interpretada, generando un fenotipo (por ejemplo, un valor real) el cuál se evalúa. Si el fenotipo de ese individuo le permite ser el "ganador", entonces es al genotipo de ese individuo "ganador" al que se le hacen las operaciones de mutación.

Del ejemplo anterior, está la siguiente función:

$$Y= -1.78*seno(X)^2 + 6.40*coseno(X)^2 + 3.07*seno(X)^3$$

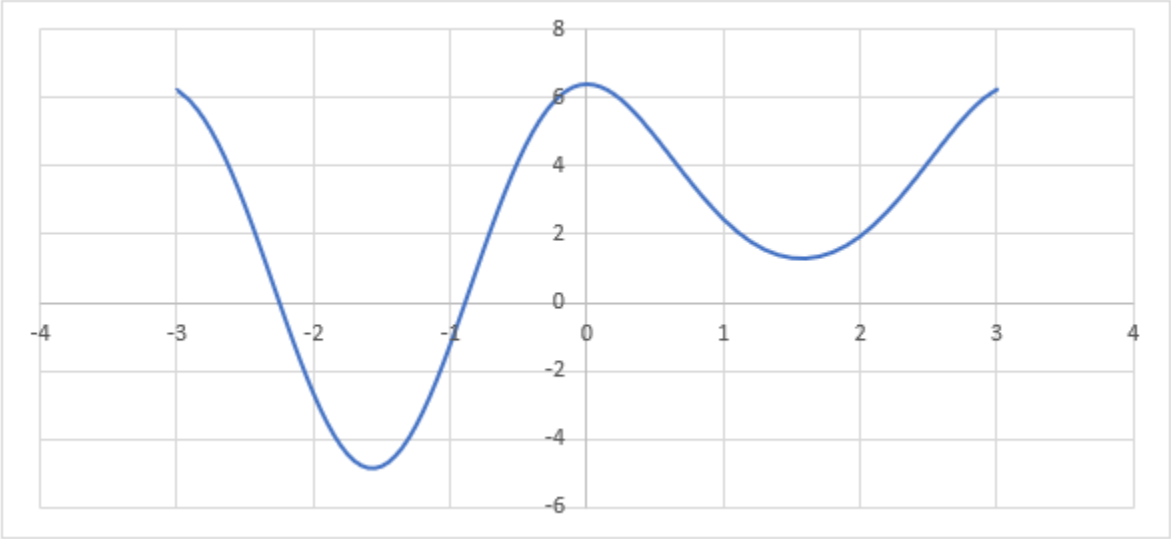

Ilustración 7: Gráfico de una ecuación

Se pide encontrar el valor de "x" entre -3 y 2, donde se obtenga el mínimo valor de "y". A ojo, viendo la gráfica es posible afirmar que el valor aproximado para x es -1.6, pero eso no es preciso. Se requiere precisión y este sería el proceso:

Paso 1

Se define que tan largo será el genotipo. Es una cadena de números binarios, entre más cifras, más preciso serán los resultados, pero tomará más tiempo en procesamiento. Una cadena puede ser: 011101101

Para el ejemplo, el valor mínimo -3 será representado por "000000000" (9 cifras) y el valor máximo 2 será representado por "111111111". ¿Cuántas combinaciones hay? Es 2^{total_cifras}, luego en este caso concreto sería 2^9 = 512. ¿Qué significa eso? Que se va a dividir en 512 partes iguales el rango entre -3 y 2, y es en una de esas partes, donde estará el valor de "x" que se obtiene el mínimo valor de "y".

Nota aclaratoria: El rango mínimo sin importar su valor será representado como "000000000" y el rango máximo sin importar su valor será representado como "111111111". ¡No se debe hacer la conversión valor binario a valor decimal aquí!

Paso 2

Interpretando el genotipo. Si -3 será "000000000" y 2 será "111111111", entonces ¿Qué sería, por ejemplo, "011011101"? ¿A qué valor x correspondería? Así se calcula:

$$x = Xmin + binarioadecimal * \frac{Xmax - Xmin}{2^{TotalCifras} - 1}$$

Probando con "000000000" (literal es un 0 en decimal)

$$x = -3 + 0 * \frac{2 - -3}{2^9 - 1} = -3 + 0 * \frac{5}{512 - 1} = -3 + 0 * \frac{5}{511} = -3$$

Probando con "111111111" (literal es un 511 en decimal)

$$x = -3 + 511 * \frac{2 - -3}{2^9 - 1} = -3 + 511 * \frac{5}{511} = -3 + 5 = 2$$

Probando con "011011101" (literal es un 221 en decimal)

$$x = -3 + 221 * \frac{2 - -3}{2^9 - 1} = -3 + 221 * \frac{5}{511} = -0{,}837573385518591$$

Paso 3

La representación binaria de un individuo da como ventaja la facilidad de implementar el operador mutación, porque es ir a una determinada posición de la cadena binaria y si esa tenía un "0" se cambia a "1" y viceversa. Por otro lado, la totalidad del individuo puede cambiarse porque cualquier posición de la cadena binaria está al alcance de ese operador. Por ejemplo:

Individuo	10011001110000
Posición al azar 9 que es	100110011**1**10000
Cambiando valor de esa posición	100110010**0**10000

Paso 4

Otra ventaja de la representación binaria es la facilidad de implementar el operador cruce, porque es simplemente dividir la cadena de los padres, combinar esos pedazos para dar origen al nuevo individuo. Por ejemplo:

Padre 1		Padre 2
10011001110000		01001001010011
1001100 1110000	Punto de corte al azar en posición 7	0100100 **1010011**

Hijo
10011001010011

Paso 5

¿Cómo el lenguaje de programación elegido para implementar el algoritmo evolutivo puede operar con valores binarios? ¿Tendrá operadores de fácil uso? ¿El desempeño será bueno? C# permite operaciones a nivel de bit, la cadena binaria estaría almacenada en una variable de tipo entero.

Paso 6

Definir una población. Los algoritmos evolutivos inician con una población inicial de individuos generados al azar. Siguiendo el enfoque anterior, la solución a esto es una lista de números de tipo entero que serán tratados como cadenas de bits.

El algoritmo sería el siguiente:

```
Algoritmo BuscaXparaMinimoValorY
Inicio
 Generar población de N individuos al azar (cadenas en binario)
 Inicio ciclo
  Seleccionar al azar dos individuos de esa población: A y B
  Generar Hijo cruzando los genes de A y B
  Mutar Hijo
  Evaluar valorY generado por el individuo A
  Evaluar valorY generado por el individuo B
  Evaluar valorY generado por el Hijo
  Si valorY de Hijo es mayor que valorY de A entonces Hijo reemplaza a A
  Si valorY de Hijo es mayor que valorY de B entonces Hijo reemplaza a B
 Fin ciclo
 Buscar individuo que genere mayor Y de la población
 Imprimir individuo
Fin
```

```
namespace Ejemplo {
 internal class Program {

  static void Main() {
   /* Único generador de números aleatorios para todo el programa */
   Random Azar = new();

   /* Iniciar el proceso */
   double Xini = -4;
   double Xfin = 1;
   int TotalIndividuos = 100;
   int NumeroCiclos = 10000;
   int TotalBits = 10; //Número de bits para representar cada individuo
   MaximoValor(Azar, Xini, Xfin, TotalIndividuos, NumeroCiclos, TotalBits);
  }

  /* Método que realiza el proceso de encontrar el valor máximo */
  static void MaximoValor(Random Azar, double Xini, double Xfin, int TotalIndividuos, int NumeroCiclos, int TotalBits) {

   /* Genera población inicial con individuos con valores aleatorios */
   int[] Individuos = new int[TotalIndividuos];

   for (int indiv = 0; indiv < TotalIndividuos; indiv++) {
    Individuos[indiv] = Azar.Next((int)Math.Pow(2, TotalBits));
   }

   int Contador = 0;
   double MejorPuntaje = double.MinValue;
   int MejorIndividuo = -1;
   double MejorValorX, MayorValorY;

   //El factor de conversión
```

```
double Divide = Math.Pow(2, TotalBits) - 1;
double Factor = (Xfin - Xini) / Divide;

for (int ciclos = 1; ciclos <= NumeroCiclos; ciclos++) {

 /* Seleccionar dos inviduos aleatoriamente */
 int Indice1 = Azar.Next(TotalIndividuos);
 int Indice2;
 do {
  Indice2 = Azar.Next(TotalIndividuos);
 } while (Indice2 == Indice1);

 /* Evalúa cada individuo */
 double Puntaje1 = Ecuacion(Xini + Individuos[Indice1] * Factor);
 double Puntaje2 = Ecuacion(Xini + Individuos[Indice2] * Factor);

 if (Puntaje1 > MejorPuntaje) { MejorPuntaje = Puntaje1; MejorIndividuo = Indice1; }
 if (Puntaje2 > MejorPuntaje) { MejorPuntaje = Puntaje2; MejorIndividuo = Indice2; }

 /* Genera un hijo con operador cruce */

 //En que posicion corta el genotipo de cada padre
 int Posicion = Azar.Next(sizeof(int) * 8); //Entero por los 8 bits de un byte

 //Extrae las partes de cada progenitor
 int Mascara = (1 << Posicion) - 1;
 int ParteA = Individuos[Indice1] >> Posicion;
 int ParteB = Individuos[Indice2] & Mascara;

 //Une las partes: inicial de A y la final de B
 int Hijo = (ParteA << Posicion) | ParteB;

 /* Muta el hijo */
 Mascara = 1 << Azar.Next(TotalBits);
 Hijo ^= Mascara;
```

```
      /* Evalúa el hijo */
      double PuntajeHijo = Ecuacion(Xini + Hijo * Factor);

      /* Si el hijo es mejor que algún progenitor, entonces se sobre-escribe el progenitor */
      if (PuntajeHijo > Puntaje1)
       Individuos[Indice1] = Hijo;

      if (PuntajeHijo > Puntaje2)
       Individuos[Indice2] = Hijo;

      /* Incrementar el contador e informar cada 1000 intentos */
      Contador++;
      if (Contador % 1000 == 0) {
       MejorValorX = Xini + Individuos[MejorIndividuo] * Factor;
       MayorValorY = Ecuacion(MejorValorX);
       Console.WriteLine($"Intento: {Contador:N0} Mejor individuo: [{MejorValorX}] con Valor: [{MayorValorY}]");
      }
     }

     MejorValorX = Xini + Individuos[MejorIndividuo] * Factor;
     MayorValorY = Ecuacion(MejorValorX);
     Console.WriteLine($"Intento: {Contador:N0} Mejor individuo: [{MejorValorX}] con Valor: [{MayorValorY}]");
    }

    static double Ecuacion(double x) {
     return 0.1 * Math.Pow(x, 6) + 0.6 * Math.Pow(x, 5) + (-0.9 * Math.Pow(x, 4)) - 6.2 * Math.Pow(x, 3) + 2 * x * x + 5 * x - 1;
    }
   }
}
```

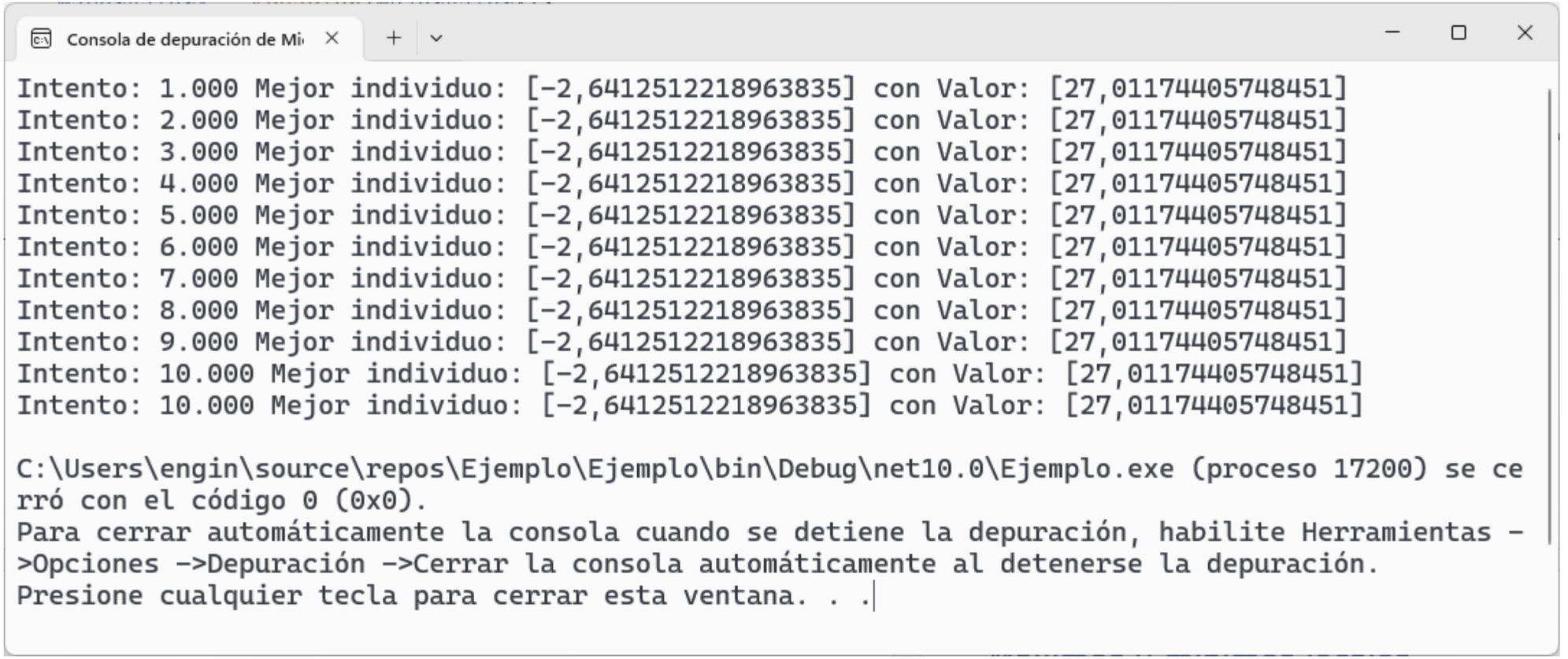

Ilustración 8: Operador cruce y mutación

Máximos y mínimos locales

En el siguiente gráfico se puede apreciar visualmente un problema con los algoritmos evolutivos. Se busca obtener el valor de X con el que se obtiene el mayor valor de Y. Los círculos rellenos representan diferentes individuos generados al azar dentro de la población.

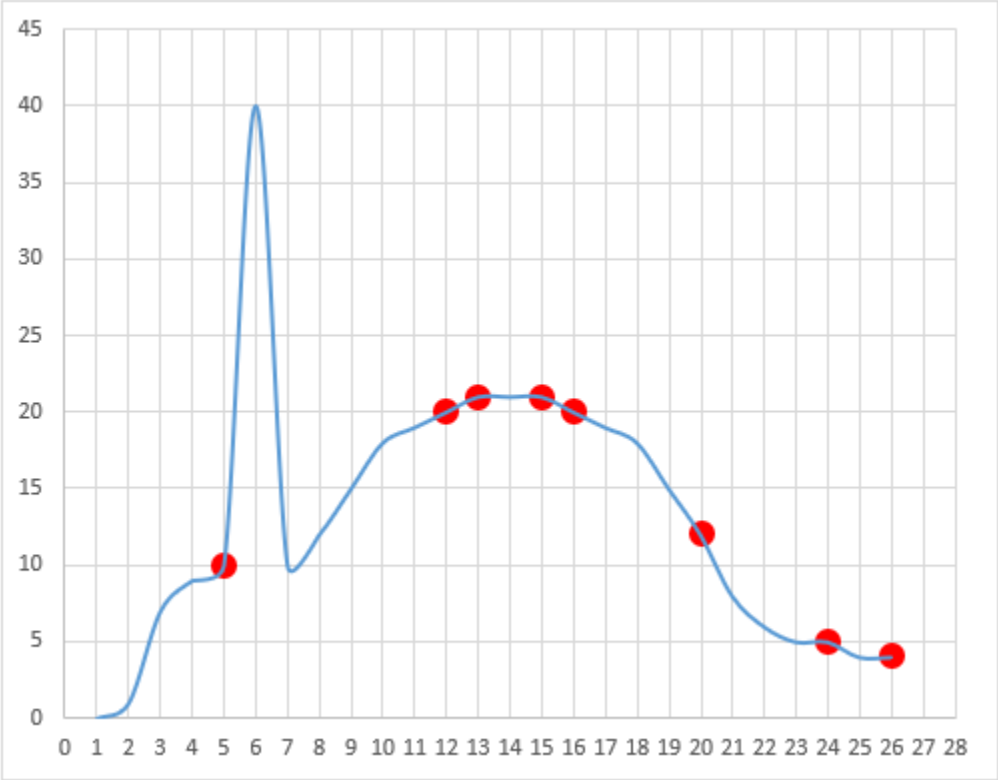

Ilustración 9: Gráfico matemático

Visualmente, el máximo "Y" se encuentra entre los valores de 5<=X<=7. Hay muchos individuos entre 11<=X<=17 que al competir contra el único individuo en la posición X=5, ganarían y se reproducirían eliminando de la población al individuo que en verdad estaba cerca de la solución global. El algoritmo evolutivo dará más puntaje a los individuos entre 11<=X<=17, y deduce erróneamente que el máximo está entre 11<=X<=17. Eso se le conoce como máximo local y una vez que el algoritmo evolutivo privilegie a los individuos de ese máximo local, se borrará cualquier esperanza de que se llegue al máximo global real.

¿Cómo solucionarlo? La variedad de los individuos dentro de la población debe mantenerse, pero no es fácil porque el mismo algoritmo premia a los ganadores con reproducirse y a los perdedores los castiga con la eliminación, así al perdedor le faltase muy poco para dar con la respuesta correcta.

El mismo problema se tendría con un mínimo local y un mínimo global.

Técnicas para evitar caer en máximos y mínimos locales:

1. Ejecutar varias veces el algoritmo evolutivo.

2. Tener varias poblaciones separadas.
3. Cada cierto número de ciclos generar aleatoriamente algunos individuos que reemplacen a algunos de la población existente.
4. Hacer uso del operador cruce que asemeja a la reproducción sexual, la cual es una técnica utilizada en la naturaleza para mantener la variabilidad.

Buscar el mayor valor en una ecuación de múltiples variables

Si hay una ecuación del tipo:

$$Y = F(a, b, c, d, e)$$

Donde a, b, c, d, e son variables independientes. ¿Cómo encontrar el mayor valor de Y si nos dan un rango en el que oscilan esas variables independientes? Este es un problema sencillo de solucionar, pero difícil de llevarlo a la práctica, porque se tendrían los siguientes ciclos anidados:

```
for (a = Minimo; a <= Maximo; a += 0.001)
 for (b = Minimo; b <= Maximo; b += 0.001)
  for (c = Minimo; c <= Maximo; c += 0.001)
   for (d = Minimo; d <= Maximo; d += 0.001)
    for (e = Minimo; e <= Maximo; e += 0.001)
```

Al terminar de ejecutar se obtendría el mayor valor de Y, pero haciendo cuentas, suponiendo que cada ciclo itera unas 1000 veces, se tiene que el total de iteraciones es 1000^5 = 1.000.000.000.000.000 de veces, una cantidad enorme que puede consumir mucho tiempo. ¿Y si requiere mayor precisión? ¿Y si son más variables? En esos casos hay un gran problema.

Los algoritmos evolutivos ofrecen una solución muy buena (no perfecta) para encontrar el mayor valor, el algoritmo es el siguiente:

```
Algoritmo BuscaXparaMinimoValorY
Inicio
 Generar población de N individuos al azar (cadenas en binario)
 Inicio ciclo
  Seleccionar al azar dos individuos de esa población: A y B
  Generar Hijo cruzando los genes de A y B
  Mutar Hijo
  Evaluar valorY generado por el individuo A
  Evaluar valorY generado por el individuo B
  Evaluar valorY generado por el Hijo
  Si valorY de Hijo es mayor que valorY de A entonces Hijo reemplaza a A
  Si valorY de Hijo es mayor que valorY de B entonces Hijo reemplaza a B
 Fin ciclo
 Buscar individuo que genere mayor Y de la población
 Imprimir individuo
Fin
```

```
namespace Ejemplo {
 internal class Program {

  static void Main() {
   /* Único generador de números aleatorios para todo el programa */
   Random Azar = new();

   /* Iniciar el proceso */
   double MinValor = -10;
   double MaxValor = 10;
   int TotalIndividuos = 100;
   int NumeroCiclos = 80000;
   int TotalBits = 10; //Número de bits para representar cada individuo
   int NumeroVariables = 5; //Número de variables en la ecuación
   MaximoValor(Azar, MinValor, MaxValor, TotalIndividuos, NumeroCiclos, TotalBits, NumeroVariables);
  }

  /* Método que realiza el proceso de encontrar el valor máximo */
  static void MaximoValor(Random Azar, double MinValor, double MaxValor, int TotalIndividuos, int NumeroCiclos, int TotalBits,
int NumeroVariables) {

   /* Genera población inicial con individuos con valores aleatorios */
   int[][] Individuos = new int[TotalIndividuos][];

   for (int indiv = 0; indiv < TotalIndividuos; indiv++) {
    Individuos[indiv] = new int[NumeroVariables];
    for (int varInterna = 0; varInterna < NumeroVariables; varInterna++) {
     Individuos[indiv][varInterna] = Azar.Next((int)Math.Pow(2, TotalBits));
    }
   }

   int Contador = 0;
```

```
double MejorPuntaje = double.MinValue;
int MejorIndividuo = -1;
double MayorValorY;

//El factor de conversión
double Divide = Math.Pow(2, TotalBits) - 1;
double Factor = (MaxValor - MinValor) / Divide;

for (int ciclos = 1; ciclos <= NumeroCiclos; ciclos++) {

 /* Seleccionar dos inviduos aleatoriamente */
 int Indice1 = Azar.Next(TotalIndividuos);
 int Indice2;
 do {
  Indice2 = Azar.Next(TotalIndividuos);
 } while (Indice2 == Indice1);

 /* Evalúa cada individuo */
 double Puntaje1 = Ecuacion(Individuos[Indice1], MinValor, Factor);
 double Puntaje2 = Ecuacion(Individuos[Indice2], MinValor, Factor);

 if (Puntaje1 > MejorPuntaje) { MejorPuntaje = Puntaje1; MejorIndividuo = Indice1; }
 if (Puntaje2 > MejorPuntaje) { MejorPuntaje = Puntaje2; MejorIndividuo = Indice2; }

 /* Genera un hijo con operador cruce */
 int[] Hijo = new int[NumeroVariables];

 int PuntoCruce = Azar.Next(NumeroVariables);
 for (int varInterna = 0; varInterna < NumeroVariables; varInterna++) {
  if (varInterna <= PuntoCruce)
   Hijo[varInterna] = Individuos[Indice1][varInterna];
  else
   Hijo[varInterna] = Individuos[Indice2][varInterna];
 }

 /* Muta el hijo */
```

```
    int PuntoMutacion = Azar.Next(NumeroVariables);
    Hijo[PuntoMutacion] = Azar.Next((int)Math.Pow(2, TotalBits));

    /* Evalúa el hijo */
    double PuntajeHijo = Ecuacion(Hijo, MinValor, Factor);

    /* Si el hijo es mejor que algún progenitor, entonces se sobre-escribe el progenitor */
    if (PuntajeHijo > Puntaje1)
     Individuos[Indice1] = Hijo;

    if (PuntajeHijo > Puntaje2)
     Individuos[Indice2] = Hijo;

    /* Incrementar el contador e informar cada 1000 intentos */
    Contador++;
    if (Contador % 1000 == 0) {
     MayorValorY = Ecuacion(Individuos[MejorIndividuo], MinValor, Factor);
     Console.WriteLine($"Intento: {Contador:N0} Mejor individuo: [{MejorIndividuo}] con Valor: [{MayorValorY}]");
    }
   }

   MayorValorY = Ecuacion(Individuos[MejorIndividuo], MinValor, Factor);
   Console.WriteLine($"Intento: {Contador:N0} Mejor individuo: [{MejorIndividuo}] con Valor: [{MayorValorY}]");

   for (int varInterna = 0; varInterna < NumeroVariables; varInterna++) {
    double ValorReal = MinValor + Individuos[MejorIndividuo][varInterna] * Factor;
    Console.WriteLine($" Variable {varInterna + 1}: {ValorReal}");
   }
  }

  static double Ecuacion(int[] Variables, double MinValor, double Factor) {
   double a = MinValor + Variables[0] * Factor;
   double b = MinValor + Variables[1] * Factor;
   double c = MinValor + Variables[2] * Factor;
   double d = MinValor + Variables[3] * Factor;
   double e = MinValor + Variables[4] * Factor;
```

```
    return 0.3 * Math.Sin(a * c - d) +
      1.7 * Math.Sin(e * b + c) +
      2.8 * Math.Cos(3.1 * b - 4.4 * a) -
      3.1 * Math.Sin(a * d - e * c) +
      0.7 * Math.Cos(a + b * c - d);
   }
  }
}
```

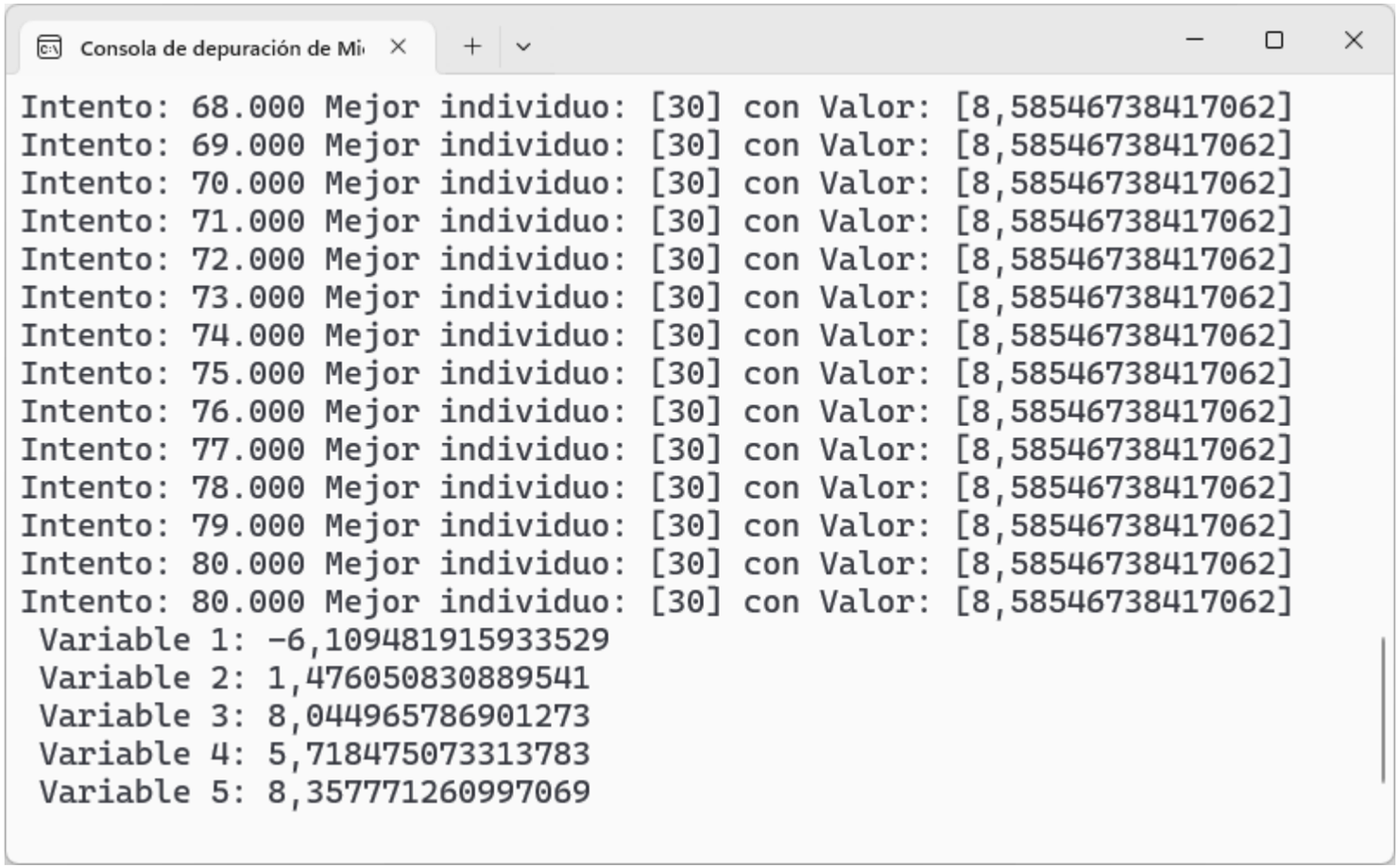

Ilustración 10: Buscar el mayor valor de Y con múltiples variables independientes

Simplificar una ecuación

Dada una ecuación original del tipo:

$$Y = a * seno(b * X + c) + d * seno(e * X + f) + \cdots + p * seno(q * X + r)$$

Como se puede observar, tiene varios bloques del tipo:

$$Y = a * seno(b * X + c)$$

¿Será posible generar una ecuación que tenga menos bloques que se acerque al comportamiento de la ecuación original? De esa forma se simplifican los cálculos y se hacen más rápido.

Se hace uso de los algoritmos evolutivos. En el ejemplo, la ecuación original tiene 30 bloques, y se busca una ecuación de 7 bloques que se acerque lo más posible a lo que hace la de 30.

```
using System.Diagnostics;

/* Algoritmo evolutivo para simplificar ecuaciones
 * Autor: Rafael Alberto Moreno Parra
 *
 * Problema:
 * Dada una ecuación compleja del tipo
 * Y = a*seno(b*X+c) + ... + p*seno(q*X+r) + ...
 * Con múltiples sumandos
 *
 * donde X es la variable independiente y
 * Y la variable dependiente, hallar una función
 * que simplifique la ecuación anterior.
 *
 * Solución:
 * Usar un algoritmo evolutivo para dar con esa
 * función que se acerque al comportamiento de la
 * función compleja.
 * ¿Cómo?
 * La ecuación compleja genera una serie de datos, luego
 * se busca la ecuación simple que se acerque a esa serie
 * de datos.
 * */


namespace Ejemplo {

 /* Cada individuo es una solución posible */
 public class Individuo {
  const double Grado_A_Radian = Math.PI / 180.0;
  public double Distancia;
  public double Puntaje;
```

```
  public double[] Coef;

  public Individuo(int TotalBloques, Random Azar, double CoefMinimo, double CoefMaximo) {
   Distancia = -1;
   Coef = new double[TotalBloques * 3];

   double Multiplica = CoefMaximo - CoefMinimo;
   for (int cont = 0; cont < Coef.Length; cont += 3) {
    Coef[cont] = Azar.NextDouble() * Multiplica + CoefMinimo;
    Coef[cont + 1] = (Azar.NextDouble() * Multiplica + CoefMinimo) * Grado_A_Radian;
    Coef[cont + 2] = (Azar.NextDouble() * Multiplica + CoefMinimo) * Grado_A_Radian;
   }
  }
}

/* Aquí se hace el proceso evolutivo */
internal class Program {

 //Valores X de entrada
 public static double[] Xentrada;

 //Valores Y generados por la ecuación
 public static double[] Yesperado;

 static void Main() {
  //Toma el tiempo
  Stopwatch temporizador = new();
  temporizador.Reset();
  temporizador.Start();

  /* Único generador de números aleatorios para todo el programa */
  Random Azar = new();

  /* Genera el dataset de la ecuación compleja */
  int BloquesDataset = 30;
  double CoefMinimo = -3;
```

```
    double CoefMaximo = 3;
    double Xmin = -720;
    double Xmax = 720;
    int TotalDatos = 100;
    GeneraDataset(Azar, BloquesDataset, CoefMinimo, CoefMaximo, Xmin, Xmax, TotalDatos);

    /* Buscar la ecuación más simple */
    int TotalIndividuos = 400;
    int NumeroCiclos = 4000000;
    int BloquesMinimo = 7;
    BuscaEcuacion(Azar, BloquesMinimo, CoefMinimo, CoefMaximo, TotalIndividuos, NumeroCiclos);

    temporizador.Stop();
    Console.WriteLine("Tiempo tomado: " + temporizador.ElapsedMilliseconds);

    /* Datos */
    Console.WriteLine("\r\nConfiguración dataset original");
    Console.WriteLine("Número de bloques originales: " + BloquesDataset);
    Console.WriteLine("Valor mínimo de los coeficientes: " + CoefMinimo);
    Console.WriteLine("Valor máximo de los coeficientes: " + CoefMaximo);
    Console.WriteLine("X mínimo: " + Xmin);
    Console.WriteLine("X máximo: " + Xmax);
    Console.WriteLine("Total datos generados: " + TotalDatos);

    Console.WriteLine("\r\nConfiguración población");
    Console.WriteLine("Total individuos: " + TotalIndividuos);
    Console.WriteLine("Número de ciclos: " + NumeroCiclos);
    Console.WriteLine("Número de bloques de cada individuo: " + BloquesMinimo);
  }

  /* Genera los datos de la ecuación compleja */
  static void GeneraDataset(Random Azar, int Bloques, double CoefMinimo, double CoefMaximo, double Xmin, double Xmax, int
TotalDatos) {
    /* Genera coeficientes aleatorios */
    Individuo Ambiente = new(Bloques, Azar, CoefMinimo, CoefMaximo);
```

```
    /* Genera valores X e Y */
    Xentrada = new double[TotalDatos];
    for (int Cont = 0; Cont < TotalDatos; Cont++)
     Xentrada[Cont] = Azar.NextDouble() * (Xmax - Xmin) + Xmin;
    Xentrada.Sort();

    Yesperado = new double[TotalDatos];
    for (int Cont = 0; Cont < TotalDatos; Cont++) {
     Yesperado[Cont] = Ecuacion(Ambiente, Xentrada[Cont]);
    }
   }

  /* Método que realiza el proceso de encontrar el valor máximo */
  static void BuscaEcuacion(Random Azar, int Bloques, double CoefMinimo, double CoefMaximo, int TotalIndividuos, int
NumeroCiclos) {

    /* Genera población inicial con coeficientes aleatorios */
    Individuo[] Individuos = new Individuo[TotalIndividuos];
    Individuo Hijo = new(Bloques, Azar, CoefMinimo, CoefMaximo);

    for (int indiv = 0; indiv < TotalIndividuos; indiv++) {
     Individuos[indiv] = new Individuo(Bloques, Azar, CoefMinimo, CoefMaximo);
    }

    double MejorDistancia = double.MaxValue;
    int MejorIndividuo = -1;
    int numCoef = Bloques * 3;

    for (int ciclos = 1; ciclos <= NumeroCiclos; ciclos++) {

     /* Seleccionar dos inviduos aleatoriamente */
     int Indice1 = Azar.Next(TotalIndividuos);
     int Indice2;
     do {
      Indice2 = Azar.Next(TotalIndividuos);
     } while (Indice2 == Indice1);
```

```
var Indiv1 = Individuos[Indice1];
var Indiv2 = Individuos[Indice2];

/* Evalúa cada individuo */
Distancia(Indiv1);
Distancia(Indiv2);

if (Indiv1.Distancia < MejorDistancia) {
 MejorDistancia = Indiv1.Distancia;
 MejorIndividuo = Indice1;
}

if (Indiv2.Distancia < MejorDistancia) {
 MejorDistancia = Indiv2.Distancia;
 MejorIndividuo = Indice2;
}

/* Genera un hijo con operador cruce (se cruzan los coeficientes) */
var coef1 = Indiv1.Coef;
var coef2 = Indiv2.Coef;
var coefHijo = Hijo.Coef;

int puntoCruce = Azar.Next(numCoef);
Array.Copy(coef1, 0, coefHijo, 0, puntoCruce + 1);
Array.Copy(coef2, puntoCruce + 1, coefHijo, puntoCruce + 1, numCoef - (puntoCruce + 1));

/* Muta el hijo */
int PuntoMutacion = Azar.Next(numCoef);
coefHijo[PuntoMutacion] += Azar.NextDouble() * 2 - 1;

/* Evalúa el hijo */
Hijo.Distancia = -1;
Distancia(Hijo);

/* Si el hijo es mejor que algún progenitor, entonces se sobre-escribe el progenitor */
```

```
      if (Hijo.Distancia < Indiv1.Distancia) {
       Array.Copy(coefHijo, coef1, coefHijo.Length);
       Indiv1.Distancia = Hijo.Distancia;
      }

      if (Hijo.Distancia < Indiv2.Distancia) {
       Array.Copy(coefHijo, coef2, coefHijo.Length);
       Indiv2.Distancia = Hijo.Distancia;
      }

      /* Informar cada 100000 intentos */
      if (ciclos % 100000 == 0) {
       Console.WriteLine($"Intento: {ciclos:N0} Mejor individuo: [{MejorIndividuo}] con Valor:
[{Individuos[MejorIndividuo].Distancia}]");
      }
     }

    Console.WriteLine($"Mejor individuo: [{MejorIndividuo}] con Valor: [{Individuos[MejorIndividuo].Distancia}]");
    for (int varInterna = 0; varInterna < Individuos[MejorIndividuo].Coef.Length; varInterna++) {
     Console.WriteLine($" Variable {varInterna + 1}: {Individuos[MejorIndividuo].Coef[varInterna]}");
    }

    Console.WriteLine("Registros");
    for (int cont = 0; cont < Xentrada.Length; cont++) {
     Console.Write(Xentrada[cont] + ";" + Yesperado[cont] + ";");
     Console.WriteLine(Ecuacion(Individuos[MejorIndividuo], Xentrada[cont]));
    }
   }

  static void Distancia(Individuo Indiv) {
   if (Indiv.Distancia != -1) return;
   Indiv.Distancia = 0;
   for (int Cont = 0; Cont < Xentrada.Length; Cont++) {
    double ValorEcuacion = Ecuacion(Indiv, Xentrada[Cont]);
    Indiv.Distancia += Math.Abs(ValorEcuacion - Yesperado[Cont]);
   }
```

```
    }

    static double Ecuacion(Individuo Indiv, double X) {
     double Acumula = 0;
     for (int bloque = 0; bloque < Indiv.Coef.Length; bloque += 3) {
      Acumula += Indiv.Coef[bloque] * Math.Sin(Indiv.Coef[bloque + 1] * X + Indiv.Coef[bloque + 2]);
     }
     return Acumula;
    }
   }
}
```

Ejemplo de ejecución:

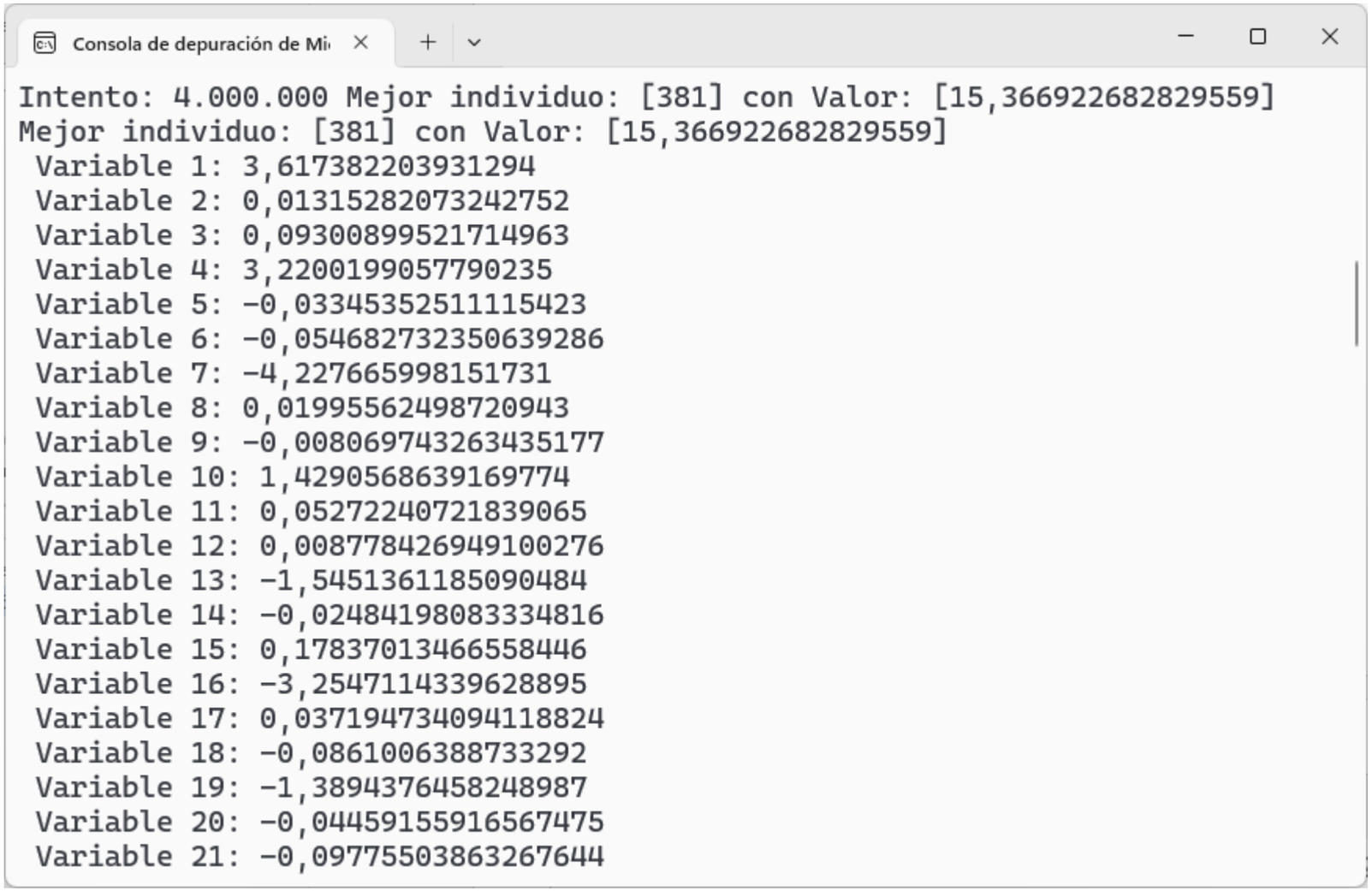

Ilustración 11: Algoritmo Evolutivo

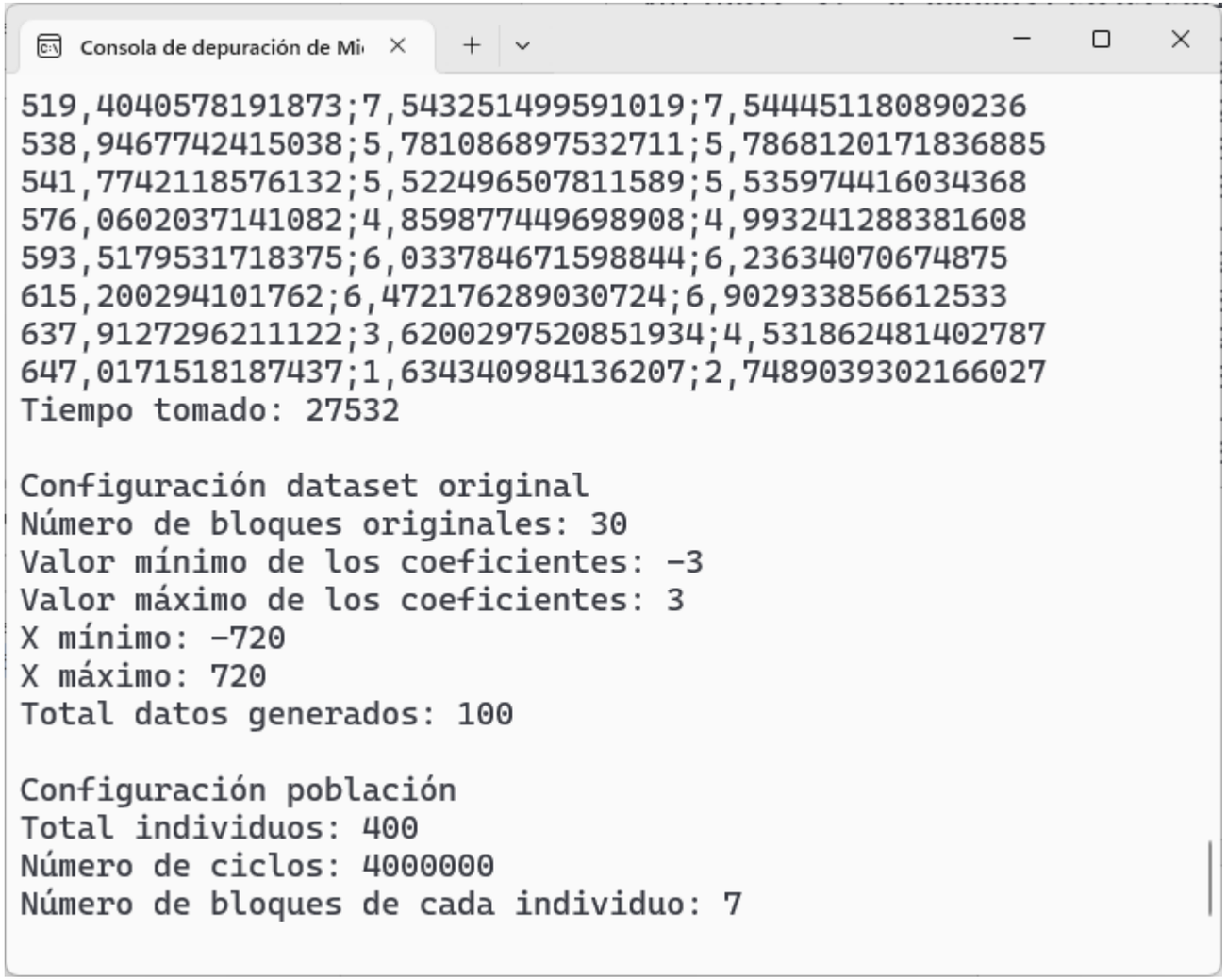

Ilustración 12: Algoritmo Evolutivo

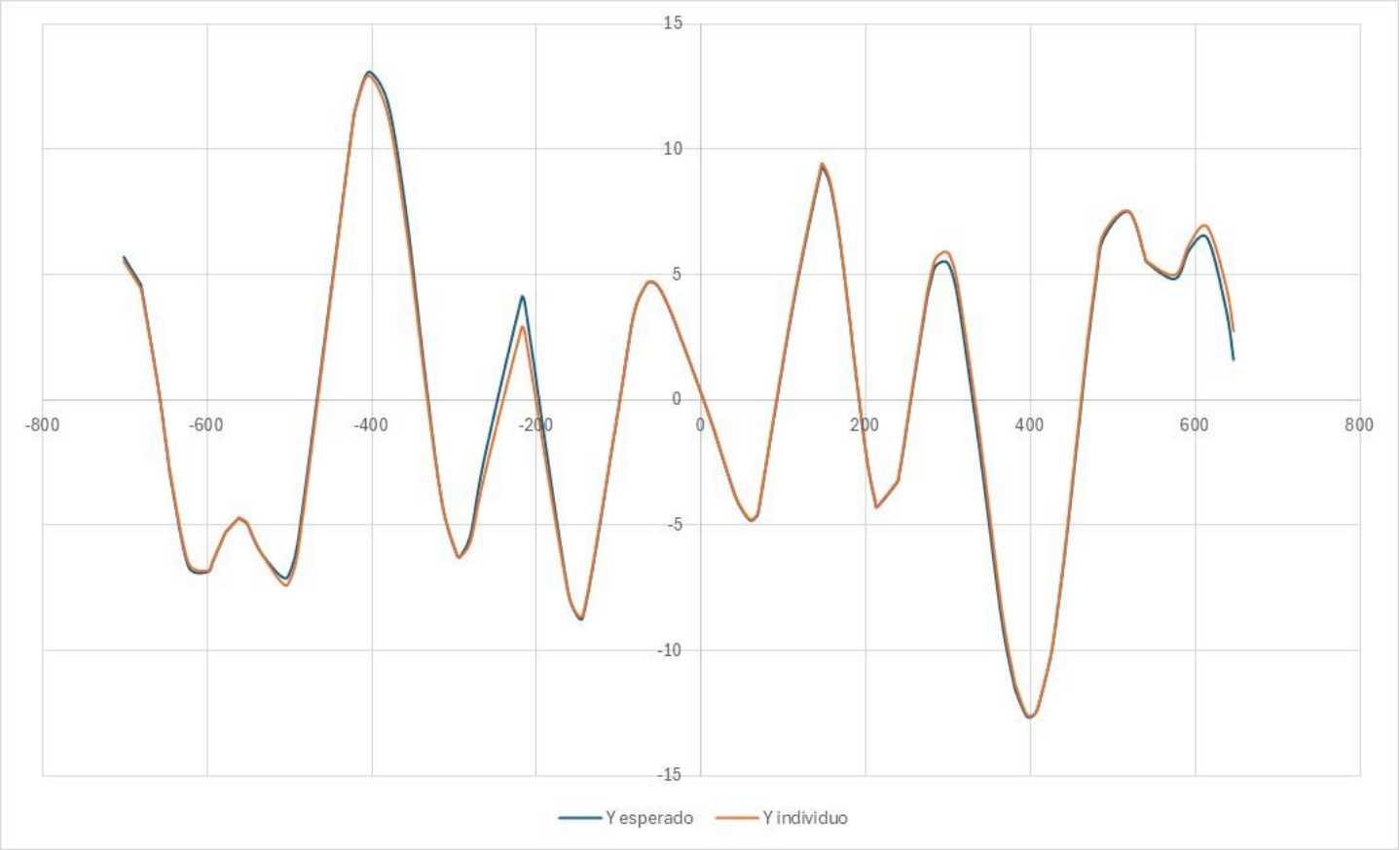

Ilustración 13: Ejemplo de simplificación de curvas

Se puede apreciar que el mejor individuo de sólo 7 bloques (es decir, 21 coeficientes), logra un comportamiento bastante similar al ambiente construido con 30 bloques (es decir, 90 coeficientes).