

C# Y .NET 8

Parte 10. Algoritmos Evolutivos

2024-07

Rafael Alberto Moreno Parra
ramsoftware@gmail.com

Contenido

Tabla de ilustraciones.....	3
Acerca del autor.....	4
Licencia de este libro	4
Licencia del software	4
Marcas registradas	5
Dedicatoria	6
Definición	7
Un ejemplo	7
Individuos y Poblaciones	10
El operador Mutación	12
El operador Cruce	19
Combinando los operadores cruce y mutación.....	25
Libertad para cambiar el algoritmo.....	29
Métricas para medir la eficacia de cada algoritmo	30
Buscar el máximo en una función	46
De genotipos y fenotipos	50
Genotipo: Operador mutación	53
Genotipo: Operador cruce.....	57
Genotipo: Operador cruce y mutación.....	61
Máximos y mínimos locales.....	65
Variando el algoritmo evolutivo	67
Tamaño de la población	67
Selección de individuos	67
Representación de los individuos.....	68
Paralelizar el algoritmo	68
Buscar el mayor valor en una ecuación de múltiples variables.....	69
Simplificar una ecuación.....	74

Tabla de ilustraciones

Ilustración 1: Algoritmo evolutivo con el operador mutación	18
Ilustración 2: Algoritmo evolutivo con el operador cruce	23
Ilustración 3: Algoritmo evolutivo: Combinando operador cruce y mutación	28
Ilustración 4: Polinomio de grado 6	46
Ilustración 5: Buscar el máximo en una función	49
Ilustración 6: Gráfico de una ecuación	50
Ilustración 7: Búsqueda del mayor valor usando genotipo de cadenas de bits	56
Ilustración 8: Genotipo: Operador cruce	60
Ilustración 9: Operador cruce y mutación	64
Ilustración 10: Gráfico matemático	65
Ilustración 11: Buscar el mayor valor de Y con múltiples variables independientes	73
Ilustración 12: Algoritmo Evolutivo	82
Ilustración 13: Ejemplo de simplificación de curvas.....	83
Ilustración 14: Ejemplo de simplificación de curvas.....	84

Acerca del autor

Rafael Alberto Moreno Parra

ramsoftware@gmail.com o enginelifemail@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: Sally, Suini, Grisú, Capuchina, Milú,
Arián, Frac y mis recordados Tinita, Tammy, Vikingo y
Michu.

Definición

Los algoritmos evolutivos, inspirados en la biología, sirven para resolver problemas que pueden llegar a ser bastante 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 algoritmos 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.

Un ejemplo

Está la siguiente cadena:

un ejemplo de cadena de caracteres

¿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.

J/000.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            Random Azar = new();

            //Imprime 10 cadenas al azar de 50 caracteres cada una
            for (int cont = 0; cont < 10; cont++)
                Console.WriteLine(CadenaAzar(Azar, 50));
        }

        //Retorna una cadena al azar
        static string CadenaAzar(Random Azar, int Longitud) {
            string Letras = "abcdefghijklmnopqrstuvwxyz ";

            string Cadena = "";
            for (int Contador = 1; Contador <= Longitud; Contador++)
                Cadena += Letras[Azar.Next(Letras.Length)];
            return Cadena;
        }
    }
}
```

En la función CadenaAzar se generan las cadenas con minúsculas y el espacio, pero no la letra Ñ o ñ. Luego es una limitante porque si la cadena original tuviese ese carácter Ñ, nunca se habría encontrado una coincidencia.

Para el algoritmo de dar con la cadena original generando cadenas al azar, este sería el código:

J/001.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Único generador de números pseudo-aleatorios
            Random Azar = new();

            //La cadena original
            string cadenaOriginal = "esta es una prueba de texto";

            //Genera cadenas al azar y ver si en algún
            //momento hay coincidencia
            int longitud = cadenaOriginal.Length;

            //Ciclo para generar cadenas al azar
            for (int Contador = 1; Contador <= 10000; Contador++) {
                string nuevaCadena = CadenaAzar(Azar, longitud);
                if (nuevaCadena == cadenaOriginal)
                    Console.WriteLine("Acertó");
            }

            Console.WriteLine("Finaliza");
        }

        //Retorna una cadena al azar
        static string CadenaAzar(Random Azar, int Longitud) {
            string Letras = "abcdefghijklmnopqrstuvwxyz ";

            string Cadena = "";
            for (int Contador = 1; Contador <= Longitud; Contador++)
                Cadena += Letras[Azar.Next(Letras.Length)];
            return Cadena;
        }
    }
}
```

No importa que tan grande ponga el ciclo (en el ejemplo, llega a 10000), es supremamente improbable que acierte la cadena generada al azar a la cadena original. En otras palabras, es muy improbable que salga el aviso "Acertó" en la ejecución del programa.

El otro enfoque, inspirado en la biología, es acercarse poco a poco a la cadena buscada.

Individuos y Poblaciones

Los individuos son las soluciones al problema, algunos individuos lo hacen mejor que otros. En el caso del ejemplo de dar con la cadena, se generan una buena cantidad de individuos (es decir cadenas al azar), algunos ni se acercan a una letra, otros, en cambio, coinciden en una letra o quizás dos. Así que se trabaja con estos últimos. Al conjunto de individuos se le conoce como Población.

¿Cómo nacen los individuos al principio? En el caso del ejemplo, son cadenas (strings) con caracteres al azar, la primera condición para este ejemplo es que la longitud de la cadena del individuo sea igual a la cadena buscada y la segunda condición es que todas las letras que conforman la cadena buscada puedan ser accesibles por los individuos.

¿Y cómo se determina la idoneidad del individuo? Cada individuo se compara con la cadena buscada, donde haya una coincidencia en letra, se le suma uno al puntaje, entre más puntaje, significa que el individuo se parece más a la cadena buscada.

¿Y luego? Durante el proceso, se seleccionan los individuos con mayor puntaje, estos son premiados, mientras que los que tienen un puntaje peor son castigados.

¿Cuál es el castigo para los perdedores? Se eliminan de la población.

¿Y cuál es el premio para los ganadores? Duplicarse y luego esta varía.

¿Cómo que el duplicado varía? Esa es la base del algoritmo evolutivo, la generación siguiente no es 100% igual a la generación anterior. Hay variación y esto se logra con dos procesos, que se les conoce como operadores: mutación y cruce.

¿Qué es eso de mutación? En ese caso, el progenitor se clona, en el ejemplo, se copia toda la cadena, luego se selecciona al azar una posición de la cadena y se reemplaza con una letra al azar. Ese nuevo individuo pasará a ocupar la vacante dejada por el individuo perdedor eliminado.

¿Qué es eso del cruce? En ese caso, se escogen dos progenitores, cada uno es una cadena distinta, luego se selecciona una posición al azar en la cadena del individuo, ese será el punto de corte, luego el hijo hereda la parte izquierda del progenitor A y la parte derecha del progenitor B. El hijo resultante se compara en puntaje con los progenitores: si el hijo supera a algún progenitor, el hijo reemplaza a ese progenitor. Puede suceder que no supere a ninguno, luego ese hijo es eliminado o puede darse el caso que supere a ambos progenitores, eliminándolos.

¿Y ya está? ¿Es aplicar esos operadores? Eso es lo más básico, de aquí se pueden hacer cambios a esos algoritmos (la imaginación es el límite) y ver cuál logra encontrar mejores soluciones, en el caso del ejemplo, dar con la cadena buscada lo más pronto posible. Por ejemplo: en el caso del cruce, en vez de dos progenitores, escoger tres, en el caso de la mutación, no sólo hacerla en una sola posición sino en varias, combinar cruce con mutación, etc..

Se ve más a fondo cada operador.

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¿pÑj¡HmfGÓ
4: ÜkJñzU;mCoQEZe! ÄmúÖN¿Ghj zoá
5: tWsLótMRFhoB;íÄpkkDgYq;LrXëÓU
6: LFopAíéÓüYj!ëÖÄvÚMycCÄíïFPLÜ?
7: ÑýérMQwlgMXaYTgúGsFfQÑIQBÜsíb
```

8: jOYnÚtAEH¿Ë;HVvPB ÄëÜáx¿fuÛch
9: ñróQEx;MqöúïEKÜiNTpRBóBWúofsö
10: ëWÄUPLúíïBVëÁÑLaómpKPÓ!İWXYÚQ

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

1: oTGBVáDgzD!óxëVoÓsbYAoVVRwkTj
2: Äh?JÄgÉmmÍ;!óQUbFtdfymNtNSjmF
3: rEVáa Z;oaÓC¿rxy?Äx¿pÑjíHmfGÓ
4: ÜkJñzU;mCoQEZe! ÄmúÖN¿Ghj zoá
5: tWsLótMRFhoB;íÄpkkDgYq;LrXëÓU
6: LFopAİéÓüYj!ëÖÄvÚMycCÄİİFPLÜ?
7: ÑyérMQwlgMXaYTgúGsFfQÑİQBÜsíb
8: jOYnÚtAEH¿Ë;HVvPB ÄëÜáx¿fuÛch
9: ñróQEx;MqöúïEKÜiNTpRBóBWúofsö
10: ëWÄUPLúíïBVëÁÑLaómpKPÓ!İWXYÚQ

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;MqöúïEKÜiNTpRBóBWúofsö 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

El individuo perdedor es eliminado de la población

1: oTGBVáDgzD!óxëVoÓsbYAoVVRwkTj
2: Äh?JÄgÉmmÍ;!óQUbFtdfymNtNSjmF
3: rEVáa Z;oaÓC¿rxy?Äx¿pÑjíHmfGÓ
4: ÜkJñzU;mCoQEZe! ÄmúÖN¿Ghj zoá
5: tWsLótMRFhoB;íÄpkkDgYq;LrXëÓU
6: LFopAİéÓüYj!ëÖÄvÚMycCÄİİFPLÜ?

7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsíb
8: jOYnÚtAEH¿Ë;HVvPB ÁëÜáx¿fuÛch
~~9: ñróQEx¿Mqöú¿EKÜ¿INTpRBóBWúofso~~
10: ëWÄUPLú¿¿BVëÁÑLaómpKPÓ!İWXYÚQ

El mejor individuo se premia creando una copia de sí mismo

3: rEVáa Z¿oaÓC¿rxy?Äx¿pÑj¿HmfGÓ

Ahora esa copia se modifica en algún punto al azar

¿?: rEVáa Z**h**oaÓC¿rxy?Äx¿pÑj¿HmfGÓ

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¿pÑj¿HmfGÓ
4: ÜkJñzU¿mCoQEZe! ÄmúÖN¿Ghj zoá
5: tWsLótMRFhoB¿¿ÄpkkDgYq¿LrXëÓU
6: LFopAİéÓüYj!ëÖÄvÚMycCÄİİFPLÜ?
7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsíb
8: jOYnÚtAEH¿Ë;HVvPB ÁëÜáx¿fuÛch
9: rEVáa ZhoaÓC¿rxy?Äx¿pÑj¿HmfGÓ
10: ëWÄUPLú¿¿BVëÁÑLaómpKPÓ!İWXYÚQ

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

```

namespace Ejemplo {

    // El individuo es una posible solución al problema
    internal class Individuo {
        public int Puntos;
        public string Cadena;

        public Individuo() {
            Puntos = -1;
            Cadena = string.Empty;
        }

        //Operador mutación
        //Cambia una letra al azar de la Cadena
        public void Muta(Random Azar, string Letras) {
            char[] Arreglo = Cadena.ToCharArray();
            int PosA = Azar.Next(Cadena.Length);
            int PosB = Azar.Next(Letras.Length);
            Arreglo[PosA] = Letras[PosB];
            Cadena = new string(Arreglo);
        }

        //Puntaje del individuo
        public void Evalua(string CadenaBusca) {
            Puntos = 0;
            for (int Cont = 0; Cont < CadenaBusca.Length; Cont++)
                if (CadenaBusca[Cont] == Cadena[Cont])
                    Puntos++;
        }
    }

    internal class Program {

        //Con que letras se va a formar los individuos
        private const string Letras = "abcdefghijklmnopqrstuvwxyz ";

        //Población: conjunto de individuos
        static List<Individuo> Pobl = [];

        static void Main() {
            Random Azar = new();

            string CadenaBusca = "prueba de algoritmos evolutivos";
            OperadorMutacion(Azar, CadenaBusca);
        }
    }
}

```

```

// Operador mutación
static void OperadorMutacion(Random Azar, string Busca) {
    int TamanoPoblacion = 350;
    int TotalCiclos = 50000;

    //Crea la población de individuos
    CreaPobl(Azar, TamanoPoblacion, Busca);

    for (int itera = 1; itera <= TotalCiclos; itera++) {

        //Selecciona dos individuos distintos al azar
        int IndivA = Azar.Next(Pobl.Count);
        int IndivB;
        do {
            IndivB = Azar.Next(Pobl.Count);
        } while (IndivA == IndivB);

        // El individuo ganador reemplaza al perdedor
        // muta esa copia y la evalúa
        if (Pobl[IndivA].Puntos > Pobl[IndivB].Puntos) {
            Pobl[IndivB].Cadena = Pobl[IndivA].Cadena;
            Pobl[IndivB].Muta(Azar, Letras);
            Pobl[IndivB].Evalua(Busca);
        }
        else if (Pobl[IndivB].Puntos > Pobl[IndivA].Puntos) {
            Pobl[IndivA].Cadena = Pobl[IndivB].Cadena;
            Pobl[IndivA].Muta(Azar, Letras);
            Pobl[IndivA].Evalua(Busca);
        }

        // Muestra resultados parciales
        if (itera % 4000 == 0) {
            List<Individuo> Temp;
            Temp = Pobl.OrderByDescending(obj => obj.Puntos).ToList();
            Console.WriteLine("Individuo: " + Temp[0].Cadena);
            Console.WriteLine(" Puntos: " + Temp[0].Puntos);
        }
    }

    //Muestra el mejor individuo con el mejor puntaje
    Console.WriteLine("\r\n\r\nFinaliza");
    Pobl = Pobl.OrderByDescending(obj => obj.Puntos).ToList();
    Console.WriteLine("Individuo: " + Pobl[0].Cadena);
    Console.WriteLine("Puntos: " + Pobl[0].Puntos);
}

// Crea la población

```



```

static void CreaPobl(Random Azar, int TotalIndiv, string Busca) {
    //Genera la población
    Pobl.Clear();
    for (int Cont = 1; Cont <= TotalIndiv; Cont++) {
        Individuo obj = new() {
            Cadena = CadAzar(Azar, Busca.Length)
        };
        obj.Evalua(Busca);
        Pobl.Add(obj);
    }
}

// Devuelve una cadena al azar
static string CadAzar(Random Azar, int Tamano) {
    string Cadena = string.Empty;
    for (int Cont = 1; Cont <= Tamano; Cont++)
        Cadena += Letras[Azar.Next(Letras.Length)].ToString();
    return Cadena;
}
}
}

```

```
Consola de depuración de Mi X + v - □ X
Individuo: ovfabxvaeorqgorxtifc ivmllmifcn Puntos: 10
Individuo: ovjeba ae lecoritgflbvvolutiq h Puntos: 16
Individuo: oideba ae aecorits sqpvolutiqos Puntos: 20
Individuo: pzdeba ae aecoritfosqnvolutioos Puntos: 22
Individuo: onueba xe aqcoritmosfpvolutivos Puntos: 24
Individuo: hrueba de aluoritmosfavolutivos Puntos: 27
Individuo: prueba de alcoritmosiavolutivos Puntos: 28
Individuo: prueba de algoritmosiavolutivos Puntos: 29
Individuo: prueba de algoritmxs evolstivos Puntos: 29
Individuo: prueba de algoritmos avolutivos Puntos: 30
Individuo: prueba de algoritmos avolutivos Puntos: 30
Individuo: prueba de algoritmos evolutivos Puntos: 31

Finaliza
Individuo: prueba de algoritmos evolutivos
Puntos: 31

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net8.0\
```

Ilustración 1: 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!óxëVoÓsbY AoVVRwkTj
2: Áh?JÄgÉmmÍ;!óQ UbFtdfymNtNSjmF
3: rEVáaqZ;oaÓC¿rxy?Äx¿pÑj íHmfGÓ
4: ÜkJñzU;mCoQEZe! ÄmúÖN¿Ghj zoá
5: tWsLótMRFhoB;íÄpkkDgYq;LrXëÓU
6: LFopAİéÓüYj!ëÖÄvÚMycCÄÍİFPLÜ?
7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsíb
8: jOYnÚtAEH¿Ë;HVvPB ÄëÜáx¿fuÜch
9: ñróQEx;MqöúİEKÜİNTpRBóBWúofsö
10: ëWÄUPLúİİBVëÄÑLaómpKPÓ!İWXYÚQ
```

Se seleccionan dos individuos al azar

```
1: oTGBVáDgzD!óxëVoÓsbY AoVVRwkTj
2: Áh?JÄgÉmmÍ;!óQ UbFtdfymNtNSjmF
3: rEVáaqZ;oaÓC¿rxy?Äx¿pÑj íHmfGÓ
4: ÜkJñzU;mCoQEZe! ÄmúÖN¿Ghj zoá
5: tWsLótMRFhoB;íÄpkkDgYq;LrXëÓU
6: LFopAİéÓüYj!ëÖÄvÚMycCÄÍİFPLÜ?
7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsíb
8: jOYnÚtAEH¿Ë;HVvPB ÄëÜáx¿fuÜch
9: ñróQEx;MqöúİEKÜİNTpRBóBWúofsö
10: ëWÄUPLúİİBVëÄÑLaómpKPÓ!İWXYÚQ
```

Esos individuos generan un tercero con la operación de cruce: Al azar se escoge un punto de corte, la primera parte la pone un individuo y el resto el otro individuo seleccionado (en fondo verde son las partes de los individuos que constituirán el nuevo individuo)

```
rEVáaqZ;oaÓC¿rxy?Äx¿pÑj íHmfGÓ y ñróQEx;MqöúİEKÜİNTpRBóBWúofsö
```

Nuevo individuo al unir las dos piezas: rEVáax;MqöüiEKÜiNTpRBóBWúofsö

Este nuevo individuo es evaluado y si es mejor que alguno de los dos escogidos anteriormente, entonces el nuevo individuo reemplaza al que les gane a esos dos progenitores. El proceso se repite varias veces hasta que se encuentra el mejor individuo que soluciona el problema.

J/003.cs

```
namespace Ejemplo {

    // El individuo es una posible solución al problema
    internal class Individuo {
        public int Puntos;
        public string Cadena;

        public Individuo() {
            Puntos = -1;
            Cadena = string.Empty;
        }

        //Operador cruce: Cruza dos cadenas en sitios al azar
        public void Cruce(Random Azar, string CadenaA, string CadenaB) {
            int Pos = Azar.Next(CadenaA.Length);

            //Cadena = IzquierdaA + DerechaB o
            //Cadena = DerechaB + IzquierdaA
            if (Azar.NextDouble() < 0.5)
                Cadena = CadenaA[..Pos] + CadenaB[Pos..];
            else
                Cadena = CadenaB[Pos..] + CadenaA[..Pos];
        }

        //Puntaje del individuo
        public void Evalua(string CadenaBusca) {
            Puntos = 0;
            for (int Cont = 0; Cont < CadenaBusca.Length; Cont++)
                if (CadenaBusca[Cont] == Cadena[Cont])
                    Puntos++;
        }
    }

    internal class Program {

        //Con que letras se va a formar los individuos
        private const string Letras = "abcdefghijklmnopqrstuvwxyz ";

        //Población: conjunto de individuos
        static List<Individuo> Pobl = [];
    }
}
```

```

static void Main() {
    Random Azar = new();

    string CadenaBusca = "prueba de algoritmos evolutivos";
    OperadorCruce(Azar, CadenaBusca);
}

// Operador mutación
static void OperadorCruce(Random Azar, string Busca) {
    int TamanoPoblacion = 500;
    int TotalCiclos = 50000;

    //Crea la población de individuos
    CreaPobl(Azar, TamanoPoblacion, Busca);

    for (int itera = 1; itera <= TotalCiclos; itera++) {

        //Selecciona dos individuos distintos al azar
        int IndivA = Azar.Next(Pobl.Count);
        int IndivB;
        do {
            IndivB = Azar.Next(Pobl.Count);
        } while (IndivA == IndivB);

        //Crea el hijo cruzando porciones de cadena de ambos padres
        Individuo Hijo = new();
        Hijo.Cruce(Azar, Pobl[IndivA].Cadena, Pobl[IndivB].Cadena);
        Hijo.Evalua(Busca);

        //Si el hijo es mejor que el primer padre, lo reemplaza
        if (Hijo.Puntos > Pobl[IndivA].Puntos) {
            Pobl[IndivA].Cadena = Hijo.Cadena;
            Pobl[IndivA].Puntos = Hijo.Puntos;
        }

        //Si el hijo es mejor que el segundo padre, lo reemplaza
        if (Hijo.Puntos > Pobl[IndivB].Puntos) {
            Pobl[IndivB].Cadena = Hijo.Cadena;
            Pobl[IndivB].Puntos = Hijo.Puntos;
        }

        // Muestra resultados parciales
        if (itera % 4000 == 0) {
            List<Individuo> Temp;
            Temp = Pobl.OrderByDescending(obj => obj.Puntos).ToList();
            Console.WriteLine("Individuo: " + Temp[0].Cadena);
            Console.WriteLine(" Puntos: " + Temp[0].Puntos);
        }
    }
}

```

```

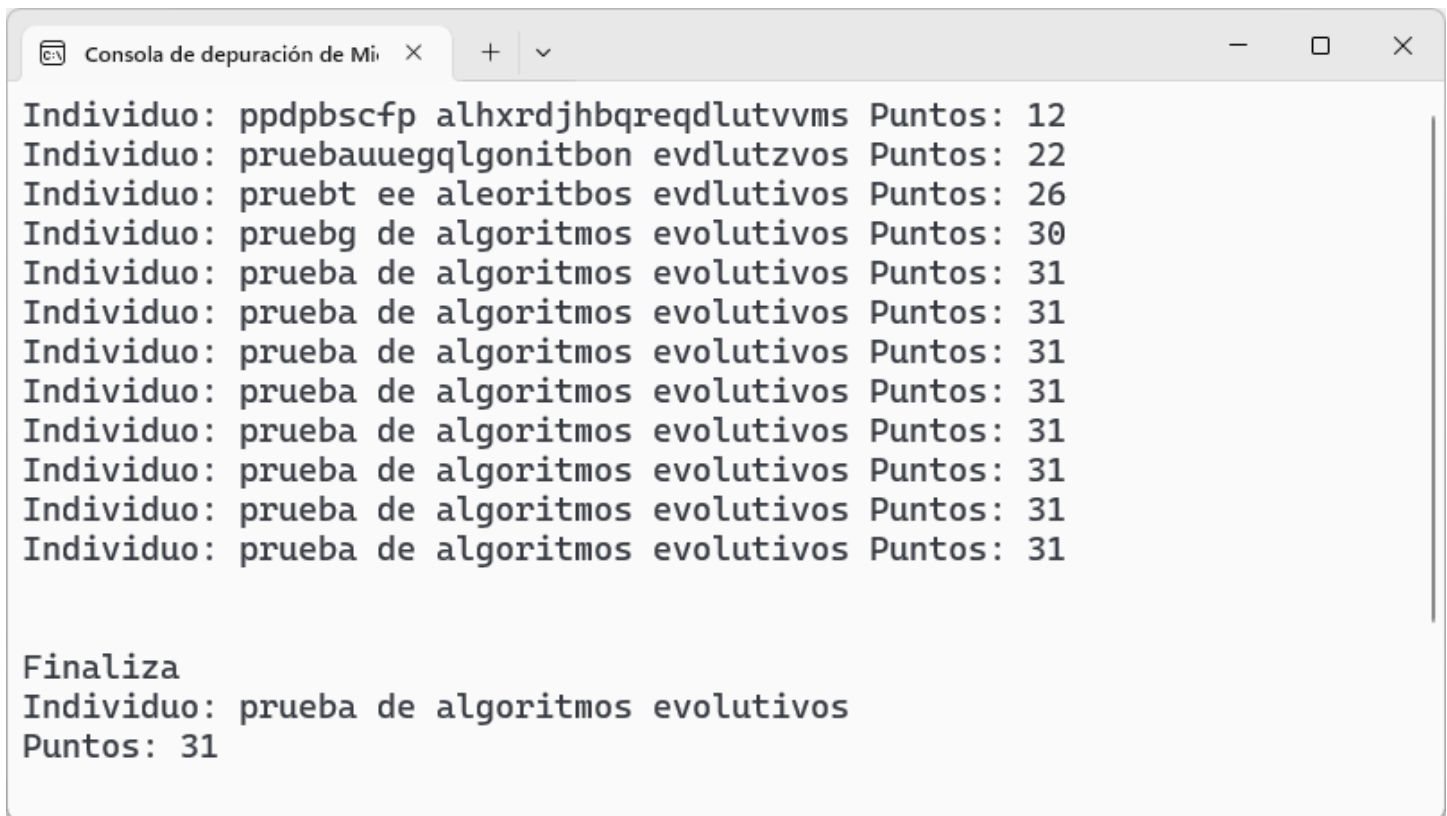
    }
}

//Muestra el mejor individuo con el mejor puntaje
Console.WriteLine("\r\n\r\nFinaliza");
Pobl = Pobl.OrderByDescending(obj => obj.Puntos).ToList();
Console.WriteLine("Individuo: " + Pobl[0].Cadena);
Console.WriteLine("Puntos: " + Pobl[0].Puntos);
}

// Crea la población
static void CreaPobl(Random Azar, int TotalIndiv, string Busca) {
    //Genera la población
    Pobl.Clear();
    for (int Cont = 1; Cont <= TotalIndiv; Cont++) {
        Individuo obj = new() {
            Cadena = CadAzar(Azar, Busca.Length)
        };
        obj.Evalua(Busca);
        Pobl.Add(obj);
    }
}

// Devuelve una cadena al azar
static string CadAzar(Random Azar, int Tamano) {
    string Cadena = string.Empty;
    for (int Cont = 1; Cont <= Tamano; Cont++)
        Cadena += Letras[Azar.Next(Letras.Length)].ToString();
    return Cadena;
}
}
}

```



```
Individuo: pppdbscfp alhxrjdjhbqreqdlutvvms Puntos: 12
Individuo: pruebauegqlgonitbon evdlutzvos Puntos: 22
Individuo: pruebt ee aleoritbos evdlutivos Puntos: 26
Individuo: pruebg de algoritmos evolutivos Puntos: 30
Individuo: prueba de algoritmos evolutivos Puntos: 31
Individuo: prueba de algoritmos evolutivos Puntos: 31
Individuo: prueba de algoritmos evolutivos Puntos: 31
Individuo: prueba de algoritmos evolutivos Puntos: 31
Individuo: prueba de algoritmos evolutivos Puntos: 31
Individuo: prueba de algoritmos evolutivos Puntos: 31
Individuo: prueba de algoritmos evolutivos Puntos: 31
Individuo: prueba de algoritmos evolutivos Puntos: 31

Finaliza
Individuo: prueba de algoritmos evolutivos
Puntos: 31
```

Ilustración 2: Algoritmo evolutivo con el operador cruce

Hay que tener cuidado con el operador cruce porque supone que la cadena buscada debe al final formarse de la unión de dos cadenas procedentes de los progenitores. Si sólo se tomase la parte izquierda del progenitor A y la derecha del progenitor B, el punto débil sería que una letra en determinada posición de la cadena buscada no existe en ninguno de los individuos de la población. Por ejemplo, si está la palabra buscada: "naranja", en la tercera posición está la letra "r", luego en toda la población debe haber individuos que tengan en la tercera posición la letra "r" o nunca va a haber acierto. Por eso, en el algoritmo aparece esta instrucción:

```
//Cadena = IzquierdaA + DerechaB o
//Cadena = DerechaB + IzquierdaA
if (Azar.NextDouble() < 0.5)
    Cadena = CadenaA[..Pos] + CadenaB[Pos..];
else
    Cadena = CadenaB[Pos..] + CadenaA[..Pos];
```

No sólo es la parte izquierda de A y la derecha de B, sino que se le puede dar la vuelta, añadiendo más variedad al individuo hijo. Sin embargo, esto genera otro problema cuando el proceso lleva ejecutando un buen tiempo porque el hijo perdería cualquier parecido a la cadena buscada. Ejemplo:

Cadena buscada: "cosecha de naranjas"

Progenitor A: "cofecha dh narpnjas"

Progenitor B: "cosecka xe nwranjas"

Hijo posible: "cofecha xe nwranjas" (se acerca más)

Hijo invierte el orden: "xe nwranjas cofecha" (se aleja mucho)

En un inicio el que un hijo invierta el orden, añade variedad a la población inicial, pero más adelante es una pérdida de tiempo.

El operador cruce es sensible al tamaño de la población, este tamaño debe ser grande para que se pueda hallar la solución requerida.

Combinando los operadores cruce y mutación

Otra estrategia es usar los dos operadores, se cruzan dos individuos y luego el hijo resultante se le aplica el operador mutación.

J/004.cs

```
namespace Ejemplo {

    // El individuo es una posible solución al problema
    internal class Individuo {
        public int Puntos;
        public string Cadena;

        public Individuo() {
            Puntos = -1;
            Cadena = string.Empty;
        }

        //Operador mutación
        //Cambia una letra al azar de la Cadena
        public void Muta(Random Azar, string Letras) {
            char[] Arreglo = Cadena.ToCharArray();
            int PosA = Azar.Next(Cadena.Length);
            int PosB = Azar.Next(Letras.Length);
            Arreglo[PosA] = Letras[PosB];
            Cadena = new string(Arreglo);
        }

        //Operador cruce: Cruza dos cadenas en sitios al azar
        public void Cruce(Random Azar, string CadenaA, string CadenaB) {
            int Pos = Azar.Next(CadenaA.Length);

            //Cadena = IzquierdaA + DerechaB o
            //Cadena = DerechaB + IzquierdaA
            if (Azar.NextDouble() < 0.5)
                Cadena = CadenaA[..Pos] + CadenaB[Pos..];
            else
                Cadena = CadenaB[Pos..] + CadenaA[..Pos];
        }

        //Puntaje del individuo
        public void Evalua(string CadenaBusca) {
            Puntos = 0;
            for (int Cont = 0; Cont < CadenaBusca.Length; Cont++)
                if (CadenaBusca[Cont] == Cadena[Cont])
                    Puntos++;
        }
    }
}
```

```

}

internal class Program {

    //Con que letras se va a formar los individuos
    private const string Letras = "abcdefghijklmnopqrstuvwxyz ";

    //Población: conjunto de individuos
    static List<Individuo> Pobl = [];

    static void Main() {
        Random Azar = new();

        string CadenaBusca = "prueba de algoritmos evolutivos";
        CruceMuta(Azar, CadenaBusca);
    }

    // Operador mutación
    static void CruceMuta(Random Azar, string Busca) {
        int TamanoPoblacion = 500;
        int TotalCiclos = 50000;

        //Crea la población de individuos
        CreaPobl(Azar, TamanoPoblacion, Busca);

        for (int itera = 1; itera <= TotalCiclos; itera++) {

            //Selecciona dos individuos distintos al azar
            int IndivA = Azar.Next(Pobl.Count);
            int IndivB;
            do {
                IndivB = Azar.Next(Pobl.Count);
            } while (IndivA == IndivB);

            //Crea el hijo cruzando porciones de cadena de ambos padres
            //y luego muta ese hijo
            Individuo Hijo = new();
            Hijo.Cruce(Azar, Pobl[IndivA].Cadena, Pobl[IndivB].Cadena);
            Hijo.Muta(Azar, Letras);
            Hijo.Evalua(Busca);

            //Si el hijo es mejor que el primer padre, lo reemplaza
            if (Hijo.Puntos > Pobl[IndivA].Puntos) {
                Pobl[IndivA].Cadena = Hijo.Cadena;
                Pobl[IndivA].Puntos = Hijo.Puntos;
            }

            //Si el hijo es mejor que el segundo padre, lo reemplaza

```

```

        if (Hijo.Puntos > Pobl[IndivB].Puntos) {
            Pobl[IndivB].Cadena = Hijo.Cadena;
            Pobl[IndivB].Puntos = Hijo.Puntos;
        }

        // Muestra resultados parciales
        if (itera % 4000 == 0) {
            List<Individuo> Temp;
            Temp = Pobl.OrderByDescending(obj => obj.Puntos).ToList();
            Console.WriteLine("Individuo: " + Temp[0].Cadena);
            Console.WriteLine(" Puntos: " + Temp[0].Puntos);
        }
    }

    //Muestra el mejor individuo con el mejor puntaje
    Console.WriteLine("\r\n\r\nFinaliza");
    Pobl = Pobl.OrderByDescending(obj => obj.Puntos).ToList();
    Console.WriteLine("Individuo: " + Pobl[0].Cadena);
    Console.WriteLine("Puntos: " + Pobl[0].Puntos);
}

// Crea la población
static void CreaPobl(Random Azar, int TotalIndiv, string Busca) {
    //Genera la población
    Pobl.Clear();
    for (int Cont = 1; Cont <= TotalIndiv; Cont++) {
        Individuo obj = new() {
            Cadena = CadAzar(Azar, Busca.Length)
        };
        obj.Evalua(Busca);
        Pobl.Add(obj);
    }
}

// Devuelve una cadena al azar
static string CadAzar(Random Azar, int Tamano) {
    string Cadena = string.Empty;
    for (int Cont = 1; Cont <= Tamano; Cont++)
        Cadena += Letras[Azar.Next(Letras.Length)].ToString();
    return Cadena;
}
}
}

```

```
Consola de depuración de Mi X + v - □ X

Individuo: zau oaldedaafgrawkdibevo nxijys Puntos: 11
Individuo: proeoa dgdaaooriafoibevo nxivvs Puntos: 17
Individuo: proekamde algoriysoy syluosvys Puntos: 20
Individuo: prueoawde algoritqos jtoluvovos Puntos: 24
Individuo: prueoawde algoritmosvhvolutsvts Puntos: 25
Individuo: pruebalde algoritmos evolukivos Puntos: 29
Individuo: pruebacde algoritmos evoyutivos Puntos: 29
Individuo: pruebacde algoritmos evolutivos Puntos: 30
Individuo: pruebacde algoritmos evolutivos Puntos: 30
Individuo: prueba de algoritmos evolutivos Puntos: 31
Individuo: prueba de algoritmos evolutivos Puntos: 31
Individuo: prueba de algoritmos evolutivos Puntos: 31

Finaliza
Individuo: prueba de algoritmos evolutivos
Puntos: 31

C:\Users\engin\source\repos\Ejemplo\Ejemplo\bin\Debug\net
8.0\Ejemplo.exe (proceso 7692) se cerró con el código 0.
```

Ilustración 3: Algoritmo evolutivo: Combinando operador cruce y mutación

Libertad para cambiar el algoritmo

La imaginación es el límite para modificar estos algoritmos, por ejemplo:

- Seleccionar tres o más individuos y que compitan entre estos para ver cuál es el mejor para generar un hijo.
- Se puede escoger sólo un individuo, duplicarlo y mutarlo, repetir hasta que la copia mutada sea mejor que el progenitor y reemplazar el progenitor.
- Mutar dos o más partes del hijo.
- Se calcularía la adaptación de toda la población y en vez de seleccionar un individuo al azar, se escoge el mejor para generar hijo y se elimina el de peor puntaje.

Métricas para medir la eficacia de cada algoritmo

¿Cómo medir la eficacia de cada idea para el algoritmo evolutivo? En un principio se podría poner el mismo número de ciclos para cada algoritmo, pero sería injusto porque un algoritmo puede hacer más cálculos que otro, luego se implementó darle el mismo tiempo en milisegundos a cada algoritmo y así probarlos.

J/005a.cs

```
namespace MetricaEvolutivo {  
    internal class Program {  
        static void Main() {  
            Estrategia simula = new Estrategia();  
            simula.Simular();  
        }  
    }  
}
```

J/005b.cs

```
namespace MetricaEvolutivo {  
    internal class Individuo {  
        public int Puntos;  
        public string Cadena;  
  
        public Individuo() {  
            Puntos = -1;  
            Cadena = string.Empty;  
        }  
  
        //Operador mutación: Cambia una letra al azar  
        public void Muta(Random Azar, string Letras) {  
            char[] Arreglo = Cadena.ToCharArray();  
            int PosA = Azar.Next(Cadena.Length);  
            int PosB = Azar.Next(Letras.Length);  
            Arreglo[PosA] = Letras[PosB];  
            Cadena = new string(Arreglo);  
        }  
  
        //Operador cruce: Cruza dos cadenas en sitios al azar  
        public void Cruce(Random Azar, string CadenaA, string CadenaB) {  
            int Pos = Azar.Next(CadenaA.Length);  
  
            //Parte izquierda de la cadena A  
            //y parte de la derecha de la cadena B  
            Cadena = CadenaA[..Pos] + CadenaB[Pos..];  
        }  
    }  
}
```

```

//Puntaje del individuo
public void Evalua(string CadenaBusca) {
    Puntos = 0;
    for (int Cont = 0; Cont < CadenaBusca.Length; Cont++)
        if (CadenaBusca[Cont] == Cadena[Cont])
            Puntos++;
    }
}
}
}

```

J/005c.cs

```

using System.Diagnostics;
using System.Text;

/* Las variables son:
 * A. Tamaño de la cadena
 * B. Tamaño de la población
 * C. Tiempo dedicado a cada estrategia
 * D. Complejidad de la cadena a buscar (simple con sólo vocales
 * a complejo con todo el abecedario)
 *
 * Como son múltiples variables, para poner a prueba
 * las diferentes estrategias, se varia sólo una variable
 * y el resto se congela */
namespace MetricaEvolutivo {
    internal class Estrategia {
        //Letras que conforman las cadenas
        private const string Letras = "aeiou";

        //Población
        private List<Individuo> Pobl = [];
        private int TotalIndividuos;

        //Único generador de números aleatorios
        private Random Azar;

        //Cadena a buscar
        private string Busca;

        //Tiempo dado a cada estrategia
        private int Tiempo;

        /* Configura la simulación */
        public void Simular() {
            Azar = new();

```

```

TotalIndividuos = 1000;
int TotalPruebas = 2; //Total para generar promedios
int TMinimo = 10; //Tiempo mínimo dado a cada estrategia
int TMaximo = 1000; //Tiempo máximo dado a cada estrategia
int TAvance = 10; //Avance de mínimo a máximo
int TCadenaBusca = 500; //Tamaño de la cadena a buscar

Console.OutputEncoding = Encoding.UTF8;
Console.WriteLine("Total de Pruebas: " + TotalPruebas);
Console.WriteLine("Tiempo mínimo: " + TMinimo);
Console.WriteLine("Tiempo máximo: " + TMaximo);
Console.WriteLine("Tamaño cadena busca: " + TCadenaBusca);
Console.WriteLine("Población: " + TotalIndividuos);

Console.WriteLine("Tiempo;E1;E2;E3;E4;E5;E6;E7");
for (Tiempo = TMinimo; Tiempo <= TMaximo; Tiempo += TAvance) {
    int AcumEstr1 = 0;
    int AcumEstr2 = 0;
    int AcumEstr3 = 0;
    int AcumEstr4 = 0;
    int AcumEstr5 = 0;
    int AcumEstr6 = 0;
    int AcumEstr7 = 0;
    for (int prueba = 1; prueba <= TotalPruebas; prueba++) {
        Busca = CadAzar(TCadenaBusca);
        AcumEstr1 += Estrategia1();
        AcumEstr2 += Estrategia2();
        AcumEstr3 += Estrategia3();
        AcumEstr4 += Estrategia4();
        AcumEstr5 += Estrategia5();
        AcumEstr6 += Estrategia6();
        AcumEstr7 += Estrategia7();
    }
    double Promedio1 = (double)AcumEstr1 / TotalPruebas;
    double Promedio2 = (double)AcumEstr2 / TotalPruebas;
    double Promedio3 = (double)AcumEstr3 / TotalPruebas;
    double Promedio4 = (double)AcumEstr4 / TotalPruebas;
    double Promedio5 = (double)AcumEstr5 / TotalPruebas;
    double Promedio6 = (double)AcumEstr6 / TotalPruebas;
    double Promedio7 = (double)AcumEstr7 / TotalPruebas;
    Console.Write(Tiempo + ";" + Promedio1);
    Console.Write("; " + Promedio2);
    Console.Write("; " + Promedio3);
    Console.Write("; " + Promedio4);
    Console.Write("; " + Promedio5);
    Console.Write("; " + Promedio6);
    Console.WriteLine("; " + Promedio7);
}

```



```

}

/* Crea la población para cada estrategia */
void CreaPoblacion() {
    //Genera la población
    Pobl.Clear();
    for (int Cont = 1; Cont <= TotalIndividuos; Cont++) {
        Individuo obj = new() {
            Cadena = CadAzar(Busca.Length)
        };
        obj.Evalua(Busca);
        Pobl.Add(obj);
    }
}

/* Devuelve una cadena al azar */
string CadAzar(int Tamano) {
    string Cadena = string.Empty;
    for (int Cont = 1; Cont <= Tamano; Cont++)
        Cadena += Letras[Azar.Next(Letras.Length)].ToString();
    return Cadena;
}

/* Estrategia 1: Operador mutación
 * */
int Estrategial() {
    CreaPoblacion();

    //Itera mientras tenga tiempo
    long T = Stopwatch.GetTimestamp();
    while (Stopwatch.GetElapsedTime(T).TotalMilliseconds <= Tiempo) {
        //Selecciona dos individuos distintos al azar
        int IndivA = Azar.Next(Pobl.Count);
        int IndivB;
        do {
            IndivB = Azar.Next(Pobl.Count);
        } while (IndivA == IndivB);

        /* El individuo ganador reemplaza al perdedor
         * muta esa copia y la evalúa */
        if (Pobl[IndivA].Puntos > Pobl[IndivB].Puntos) {
            Pobl[IndivB].Cadena = Pobl[IndivA].Cadena;
            Pobl[IndivB].Muta(Azar, Letras);
            Pobl[IndivB].Evalua(Busca);
        }
        else if (Pobl[IndivB].Puntos > Pobl[IndivA].Puntos) {
            Pobl[IndivA].Cadena = Pobl[IndivB].Cadena;
            Pobl[IndivA].Muta(Azar, Letras);
        }
    }
}

```

```

        Pobl[IndivA].Evalua (Busca);
    }
}

//Muestra el mejor individuo con el mejor puntaje
Pobl = Pobl.OrderByDescending(obj => obj.Puntos).ToList();
//Console.WriteLine("El: " + Pobl[0].Cadena + " == " + Busca + "
Puntos: " + Pobl[0].Puntos);
return Pobl[0].Puntos;
}

/* Estrategia 2:
 * Ordena del puntaje mayor a menor la población
 * Toma el primer individuo, lo copia sobre el último
 * y luego muta esa copia */
int Estrategia2() {
    CreaPoblacion();

    //Itera mientras tenga tiempo
    long T = Stopwatch.GetTimestamp();
    while (Stopwatch.GetElapsedTime(T).TotalMilliseconds <= Tiempo) {

        //Ordena la lista de mejor a peor individuo
        Pobl = [... Pobl.OrderByDescending(obj => obj.Puntos)];
        int Ultimo = Pobl.Count - 1;

        //El mejor individuo reemplaza al peor,
        //muta esa copia y la evalúa
        Pobl[Ultimo].Cadena = Pobl[0].Cadena;
        Pobl[Ultimo].Muta (Azar, Letras);
        Pobl[Ultimo].Evalua (Busca);
    }

    //Muestra el mejor individuo con el mejor puntaje
    Pobl = [... Pobl.OrderByDescending(obj => obj.Puntos)];
    return Pobl[0].Puntos;
}

/* Estrategia 3: Operador mutación.
 * Si hay empate en puntaje, entonces toma uno de los dos
 * y lo elimina, luego crea un individuo nuevo
 * */
int Estrategia3() {
    CreaPoblacion();

    //Itera mientras tenga tiempo
    long T = Stopwatch.GetTimestamp();
    while (Stopwatch.GetElapsedTime(T).TotalMilliseconds <= Tiempo) {

```

```

//Selecciona dos individuos distintos al azar
int IndivA = Azar.Next(Pobl.Count);
int IndivB;
do {
    IndivB = Azar.Next(Pobl.Count);
} while (IndivA == IndivB);

//El individuo ganador reemplaza al perdedor,
//muta esa copia y la evalúa
if (Pobl[IndivA].Puntos > Pobl[IndivB].Puntos) {
    Pobl[IndivB].Cadena = Pobl[IndivA].Cadena;
    Pobl[IndivB].Muta(Azar, Letras);
    Pobl[IndivB].Evalua(Busca);
}
else if (Pobl[IndivB].Puntos > Pobl[IndivA].Puntos) {
    Pobl[IndivA].Cadena = Pobl[IndivB].Cadena;
    Pobl[IndivA].Muta(Azar, Letras);
    Pobl[IndivA].Evalua(Busca);
}
else /* Hubo empate, luego elimina uno de los dos
    * y genera un nuevo individuo */
{
    Pobl.RemoveAt(IndivA);
    Individuo obj = new() {
        Cadena = CadAzar(Busca.Length)
    };
    obj.Evalua(Busca);
    Pobl.Add(obj);
}

//Muestra el mejor individuo con el mejor puntaje
Pobl = [... Pobl.OrderByDescending(obj => obj.Puntos)];
return Pobl[0].Puntos;
}

/* Estrategia 4: Operador cruce
* */
int Estrategia4() {
    CreaPoblacion();

    //Itera mientras tenga tiempo
    long T = Stopwatch.GetTimestamp();
    while (Stopwatch.GetElapsedTime(T).TotalMilliseconds <= Tiempo) {

        //Selecciona dos individuos distintos al azar
        int IndivA = Azar.Next(Pobl.Count);

```

```

int IndivB;
do {
    IndivB = Azar.Next(Pobl.Count);
} while (IndivA == IndivB);

//Crea el hijo cruzando porciones de cadena de ambos padres
Individuo Hijo = new();
Hijo.Cruce(Azar, Pobl[IndivA].Cadena, Pobl[IndivB].Cadena);
Hijo.Evalua(Busca);

//Si el hijo es mejor que el primer padre, lo reemplaza
if (Hijo.Puntos > Pobl[IndivA].Puntos) {
    Pobl[IndivA].Cadena = Hijo.Cadena;
    Pobl[IndivA].Puntos = Hijo.Puntos;
}

//Si el hijo es mejor que el segundo padre, lo reemplaza
if (Hijo.Puntos > Pobl[IndivB].Puntos) {
    Pobl[IndivB].Cadena = Hijo.Cadena;
    Pobl[IndivB].Puntos = Hijo.Puntos;
}
}

//Muestra el mejor individuo con el mejor puntaje
Pobl = Pobl.OrderByDescending(obj => obj.Puntos).ToList();
return Pobl[0].Puntos;
}

/* Estrategia 5: Operador cruce + mutación
* */
int Estrategia5() {
    CreaPoblacion();

    //Itera mientras tenga tiempo
    long T = Stopwatch.GetTimestamp();
    while (Stopwatch.GetElapsedTime(T).TotalMilliseconds <= Tiempo) {

        //Selecciona dos individuos distintos al azar
        int IndivA = Azar.Next(Pobl.Count);
        int IndivB;
        do {
            IndivB = Azar.Next(Pobl.Count);
        } while (IndivA == IndivB);

        //Crea el hijo cruzando porciones de cadena de ambos padres
        Individuo Hijo = new();
        Hijo.Cruce(Azar, Pobl[IndivA].Cadena, Pobl[IndivB].Cadena);
    }
}

```

```

//Muta el hijo además
Hijo.Muta(Azar, Letras);

Hijo.Evalua(Busca);

//Si el hijo es mejor que el primer padre, lo reemplaza
if (Hijo.Puntos > Pobl[IndivA].Puntos) {
    Pobl[IndivA].Cadena = Hijo.Cadena;
    Pobl[IndivA].Puntos = Hijo.Puntos;
}

//Si el hijo es mejor que el segundo padre, lo reemplaza
if (Hijo.Puntos > Pobl[IndivB].Puntos) {
    Pobl[IndivB].Cadena = Hijo.Cadena;
    Pobl[IndivB].Puntos = Hijo.Puntos;
}
}

//Muestra el mejor individuo con el mejor puntaje
Pobl = [.. Pobl.OrderByDescending(obj => obj.Puntos)];
return Pobl[0].Puntos;
}

/* Estrategia 6: Cada individuo de una población
 * mejora poco a poco
 * */
int Estrategia6() {
    CreaPoblacion();

    int Individuo = 0;

    //Itera mientras tenga tiempo
    long T = Stopwatch.GetTimestamp();
    while (Stopwatch.GetElapsedTime(T).TotalMilliseconds <= Tiempo) {
        string CadAntes = Pobl[Individuo].Cadena;
        int PuntajeAntes = Pobl[Individuo].Puntos;

        Pobl[Individuo].Muta(Azar, Letras);
        Pobl[Individuo].Evalua(Busca);

        //Si la nueva mutación desmejora el individuo
        //entonces restaura la cadena anterior
        if (Pobl[Individuo].Puntos < PuntajeAntes) {
            Pobl[Individuo].Cadena = CadAntes;
            Pobl[Individuo].Puntos = PuntajeAntes;
        }

        if (++Individuo >= Pobl.Count) Individuo = 0;
    }
}

```

```

}

//Muestra el mejor individuo con el mejor puntaje
Pobl = [... Pobl.OrderByDescending(obj => obj.Puntos)];
return Pobl[0].Puntos;
}

/* Estrategia 7: Mutar dos veces
 * */
int Estrategia7() {
    CreaPoblacion();

    //Itera mientras tenga tiempo
    long T = Stopwatch.GetTimestamp();
    while (Stopwatch.GetElapsedTime(T).TotalMilliseconds <= Tiempo) {
        //Selecciona dos individuos distintos al azar
        int IndivA = Azar.Next(Pobl.Count);
        int IndivB;
        do {
            IndivB = Azar.Next(Pobl.Count);
        } while (IndivA == IndivB);

        /* El individuo ganador reemplaza al perdedor
         * muta esa copia y la evalúa */
        if (Pobl[IndivA].Puntos > Pobl[IndivB].Puntos) {
            Pobl[IndivB].Cadena = Pobl[IndivA].Cadena;
            Pobl[IndivB].Muta(Azar, Letras);
            Pobl[IndivB].Muta(Azar, Letras);
            Pobl[IndivB].Evalua(Busca);
        }
        else if (Pobl[IndivB].Puntos > Pobl[IndivA].Puntos) {
            Pobl[IndivA].Cadena = Pobl[IndivB].Cadena;
            Pobl[IndivA].Muta(Azar, Letras);
            Pobl[IndivA].Muta(Azar, Letras);
            Pobl[IndivA].Evalua(Busca);
        }
    }

    //Muestra el mejor individuo con el mejor puntaje
    Pobl = Pobl.OrderByDescending(obj => obj.Puntos).ToList();
    return Pobl[0].Puntos;
}
}
}

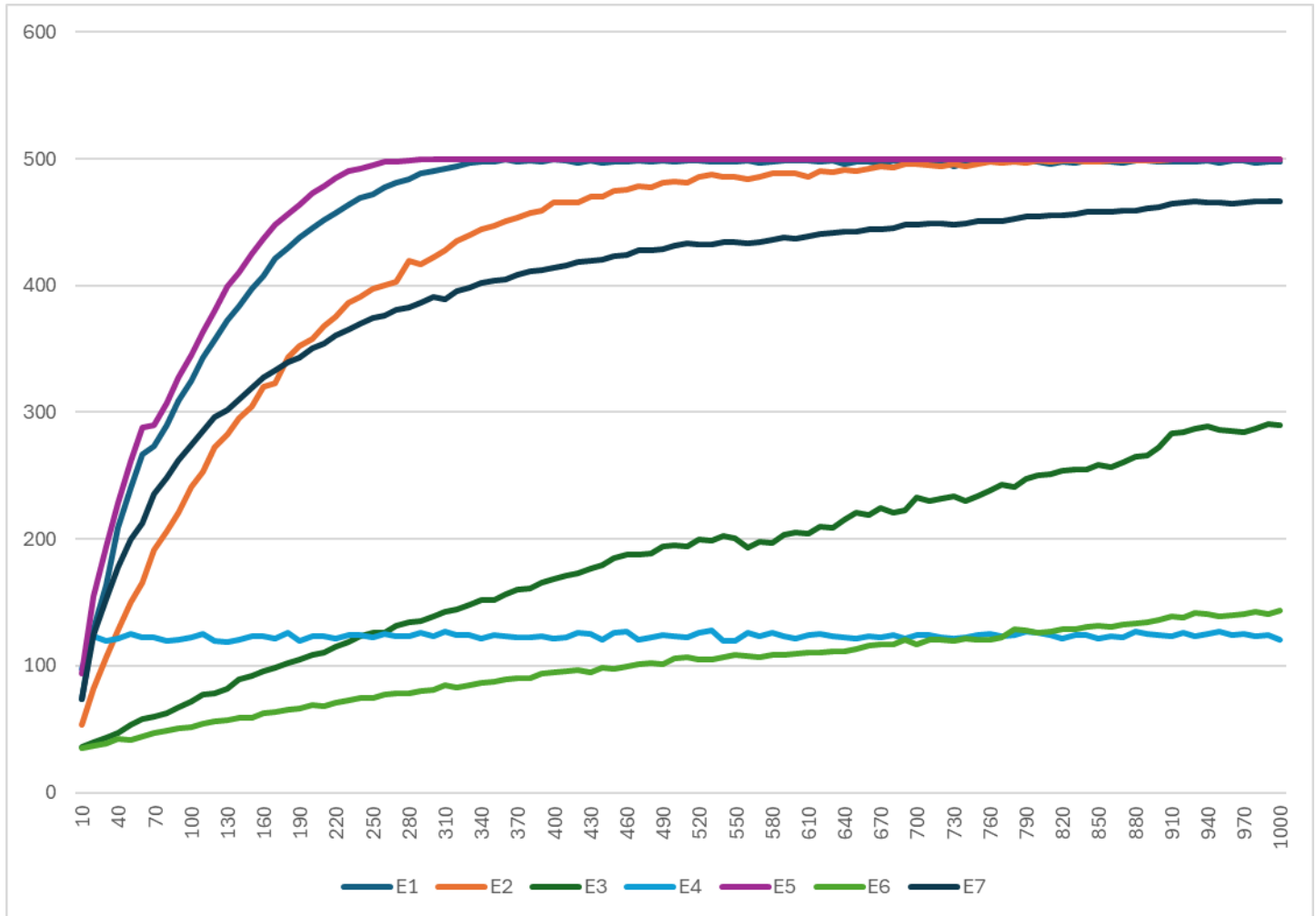
```

Las estrategias implementadas fueron:

E1	Operador mutación.
E2	Ordena del puntaje mayor a menor la población. Toma el primer individuo, lo copia sobre el último y luego muta esa copia.
E3	Operador mutación. Si hay empate en puntaje entre los dos individuos seleccionados al azar, entonces toma uno de los dos y lo elimina, luego crea un individuo nuevo.
E4	Operador cruce clásico. Toma la parte izquierda del padre A y la parte derecha del padre B. No voltea la cadena.
E5	Operador cruce clásico + mutación
E6	Cada individuo de una población mejora poco a poco
E7	Mutar dos veces

Tamaño cadena a buscar: 500 caracteres

Población: 400

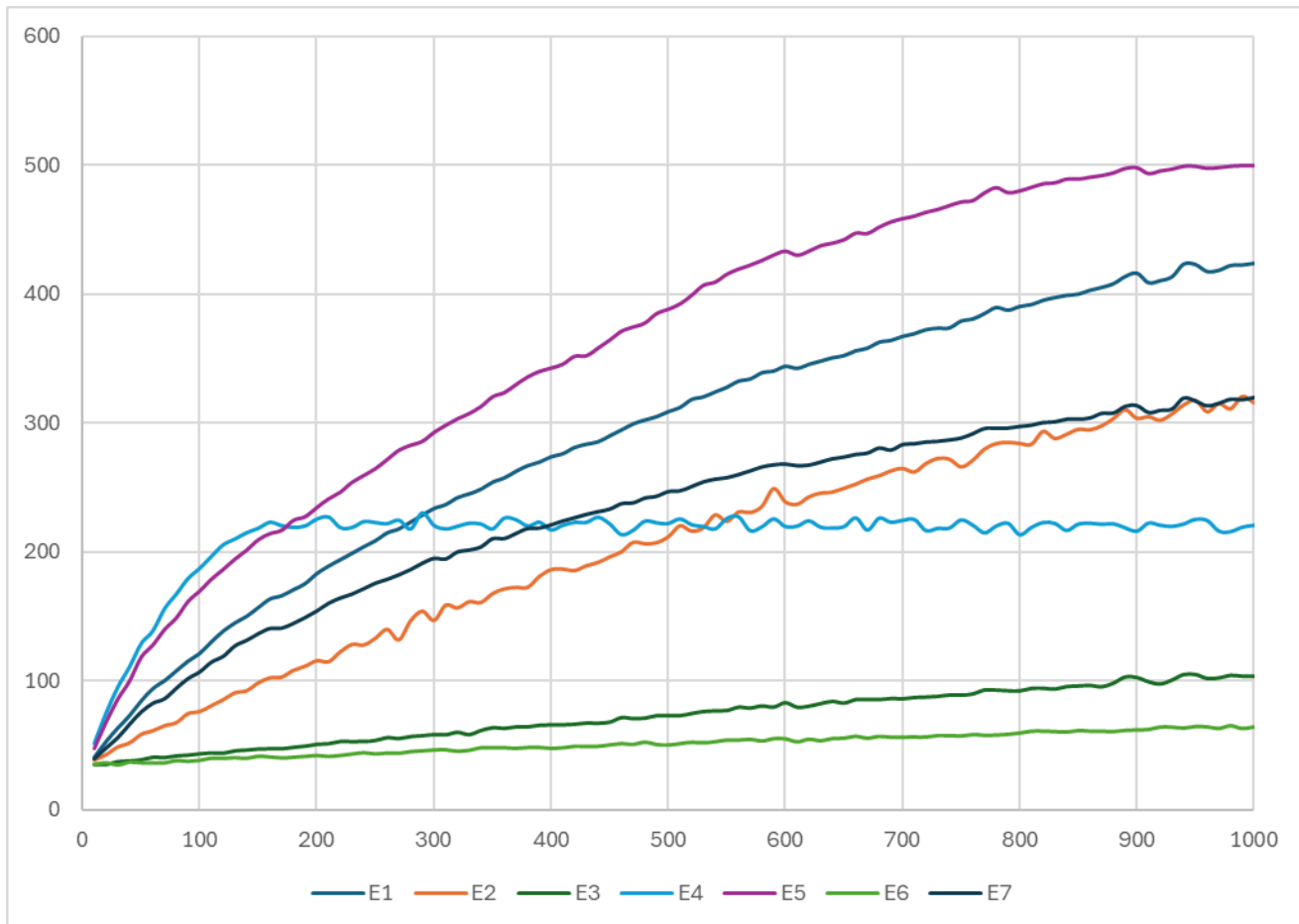


De la más eficiente a la menos eficiente

E5	Operador cruce clásico + mutación
E1	Operador mutación.
E2	Ordena del puntaje mayor a menor la población. Toma el primer individuo, lo copia sobre el último y luego muta esa copia.
E7	Mutar dos veces
E3	Operador mutación. Si hay empate en puntaje entre los dos individuos seleccionados al azar, entonces toma uno de los dos y lo elimina, luego crea un individuo nuevo.
E6	Cada individuo de una población mejora poco a poco
E4	Operador cruce clásico. Toma la parte izquierda del padre A y la parte derecha del padre B. No voltea la cadena.

Tamaño cadena busca: 500

Población: 2.000 (dos mil)

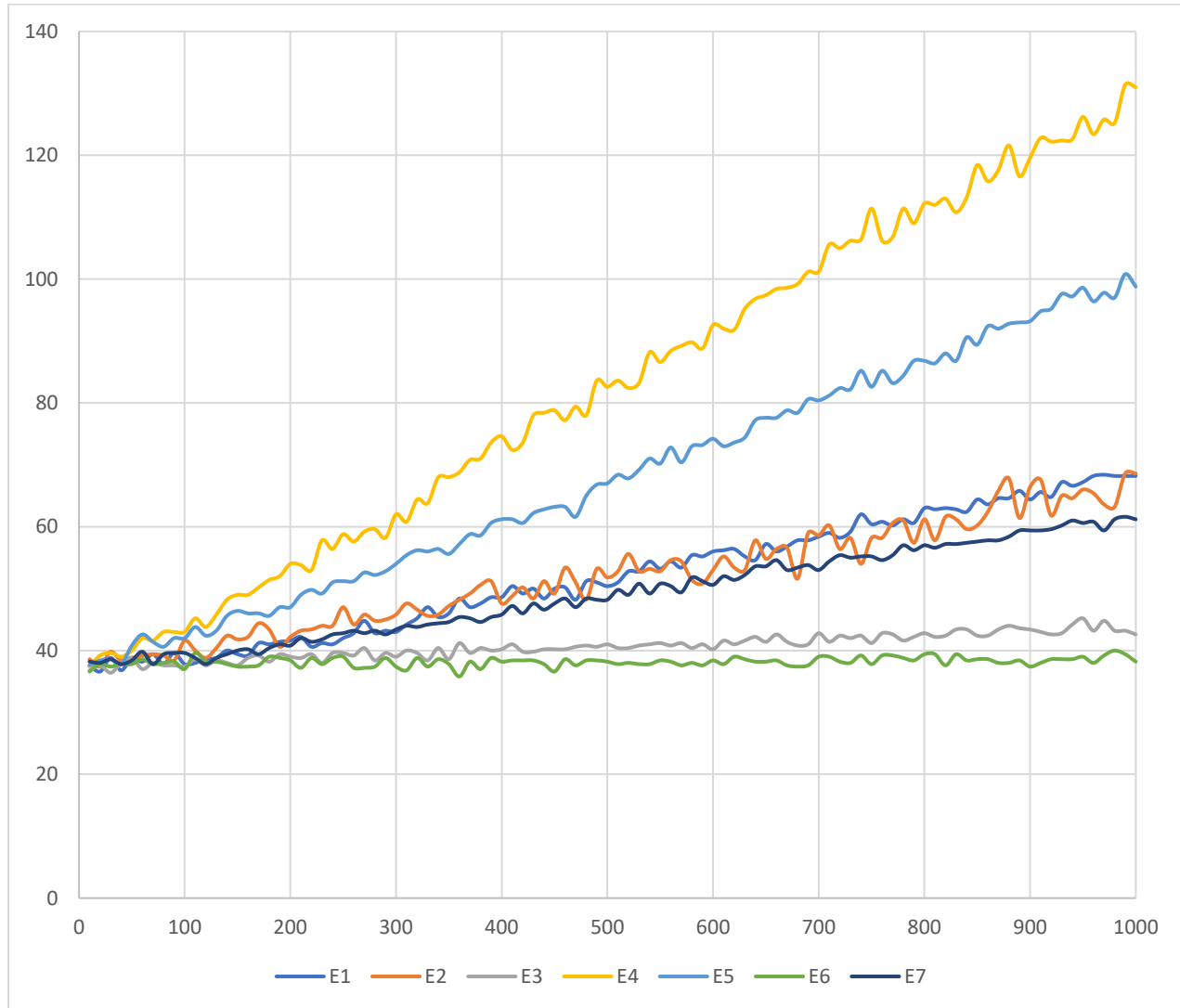


De la más eficiente a la menos eficiente

E5	Operador cruce clásico + mutación
E1	Operador mutación.
E7	Mutar dos veces
E2	Ordena del puntaje mayor a menor la población. Toma el primer individuo, lo copia sobre el último y luego muta esa copia.
E4	Operador cruce clásico. Toma la parte izquierda del padre A y la parte derecha del padre B. No voltea la cadena.
E3	Operador mutación. Si hay empate en puntaje entre los dos individuos seleccionados al azar, entonces toma uno de los dos y lo elimina, luego crea un individuo nuevo.
E6	Cada individuo de una población mejora poco a poco

Tamaño cadena busca: 500

Población: 20.000 (veinte mil)



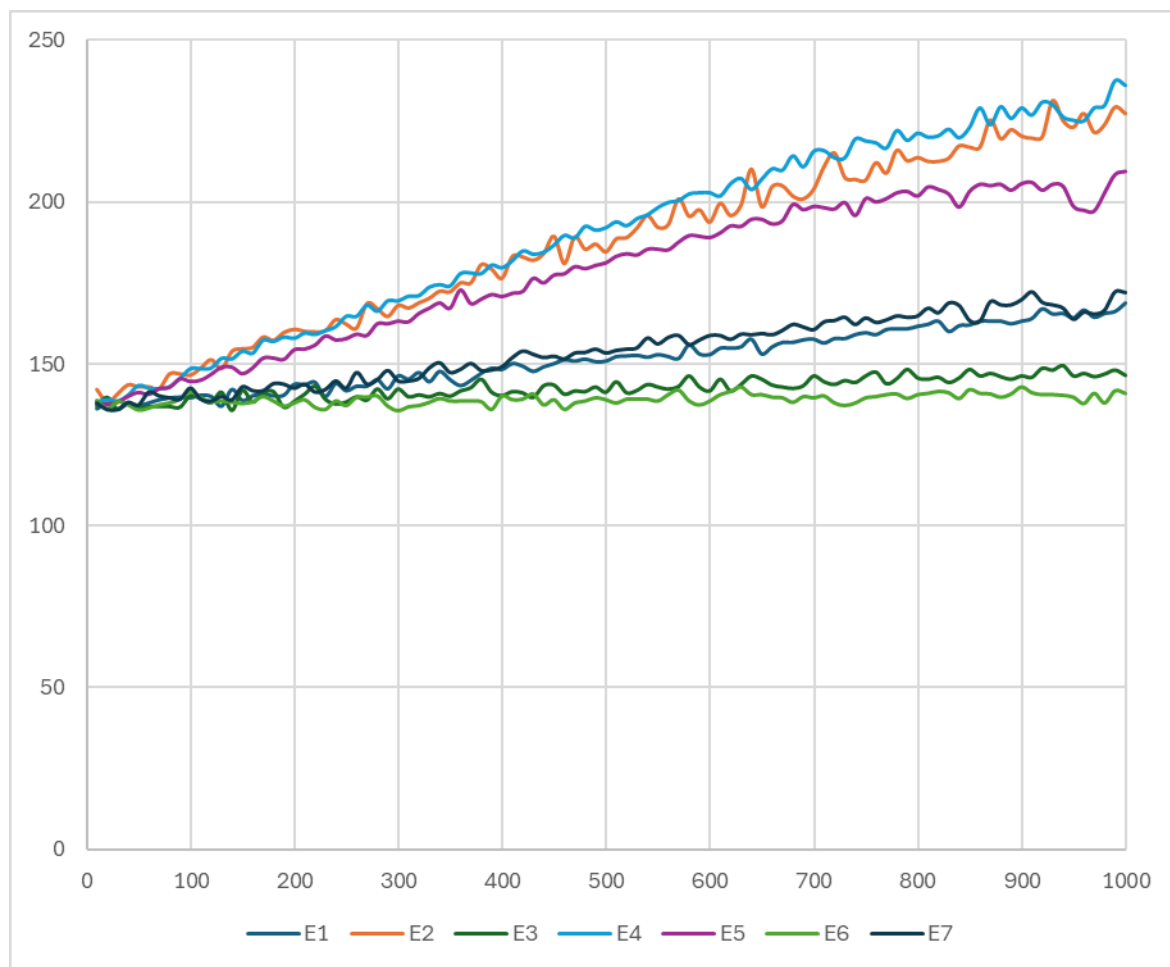
De la más eficiente a la menos eficiente

E4	Operador cruce clásico. Toma la parte izquierda del padre A y la parte derecha del padre B. No voltea la cadena.
E5	Operador cruce clásico + mutación
E2	Ordena del puntaje mayor a menor la población. Toma el primer individuo, lo copia sobre el último y luego muta esa copia.
E1	Operador mutación.
E7	Mutar dos veces
E3	Operador mutación. Si hay empate en puntaje entre los dos individuos seleccionados al azar, entonces toma uno de los dos y lo elimina, luego crea un individuo nuevo.
E6	Cada individuo de una población mejora poco a poco

Disminuyendo la complejidad de la cadena a buscar, sólo son las vocales, no consonantes, ni espacios.

Tamaño cadena busca: 500

Población: 20.000 (veinte mil)

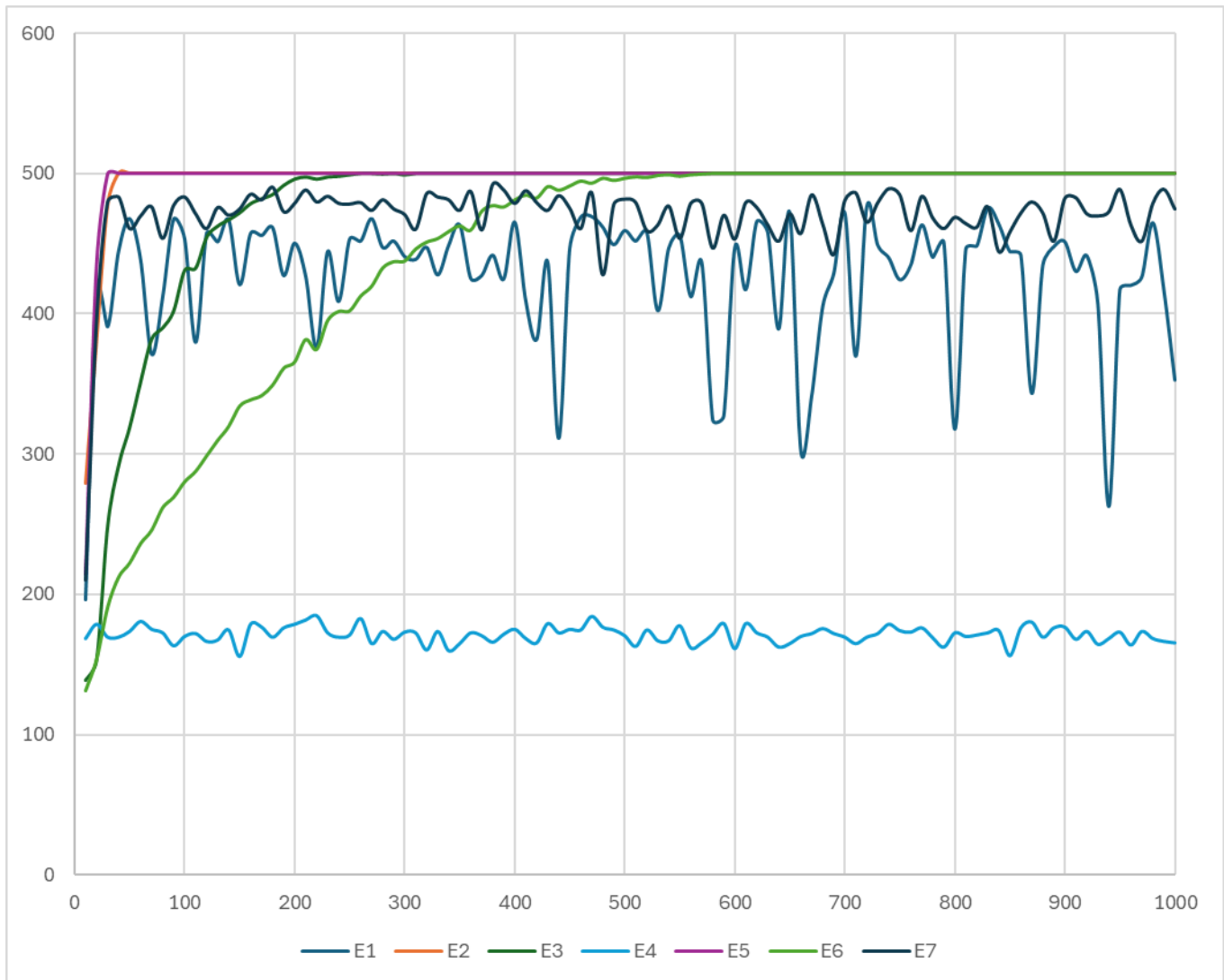


De la más eficiente a la menos eficiente

E4	Operador cruce clásico. Toma la parte izquierda del padre A y la parte derecha del padre B. No voltea la cadena.
E2	Ordena del puntaje mayor a menor la población. Toma el primer individuo, lo copia sobre el último y luego muta esa copia.
E5	Operador cruce clásico + mutación
E7	Mutar dos veces
E1	Operador mutación.
E3	Operador mutación. Si hay empate en puntaje entre los dos individuos seleccionados al azar, entonces toma uno de los dos y lo elimina, luego crea un individuo nuevo.
E6	Cada individuo de una población mejora poco a poco

Tamaño cadena busca: 500 (solo vocales)

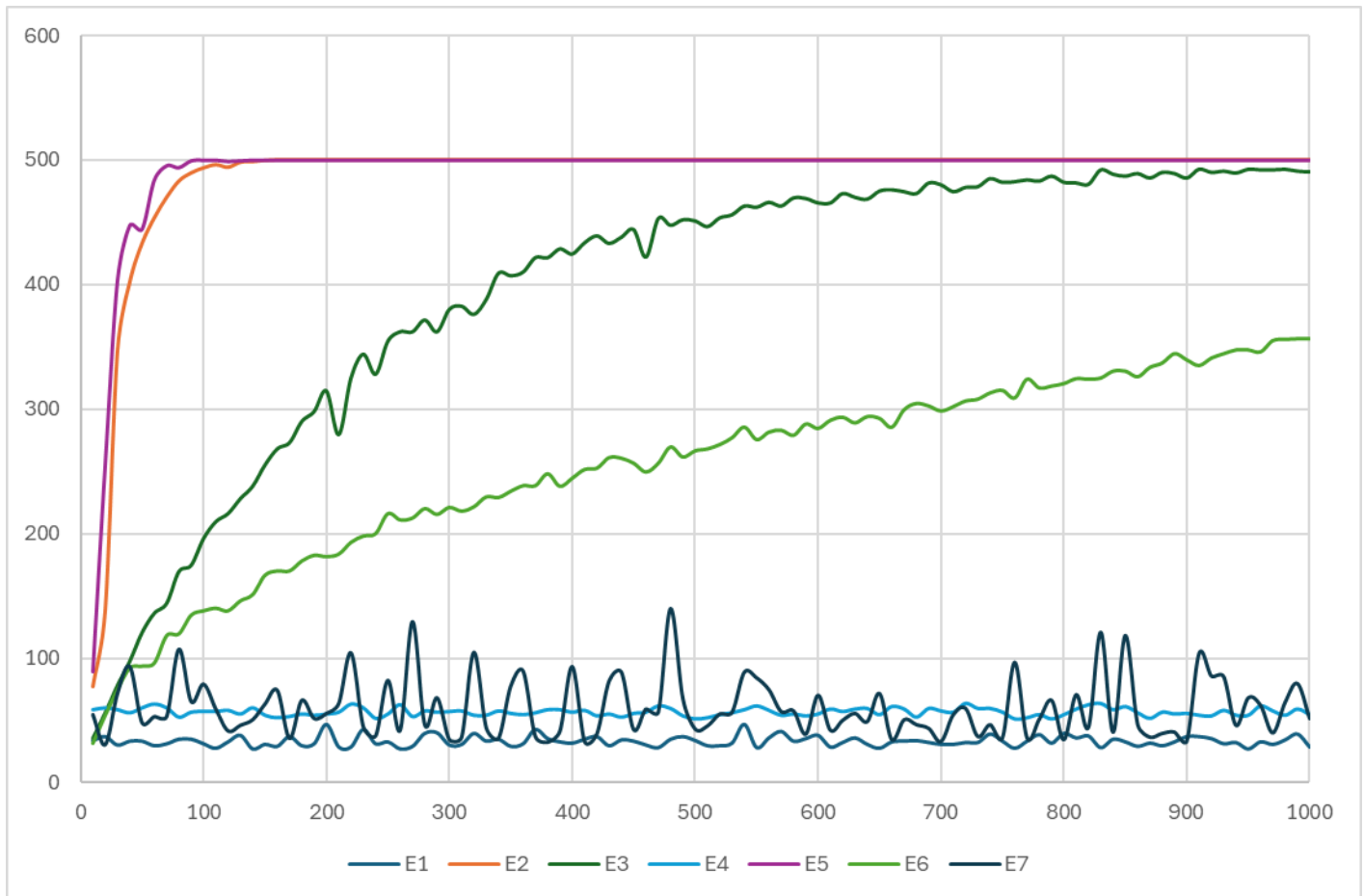
Población: 50 (población pequeña)



E2	Ordena del puntaje mayor a menor la población. Toma el primer individuo, lo copia sobre el último y luego muta esa copia. Máximo puntaje.
E3	Operador mutación. Si hay empate en puntaje entre los dos individuos seleccionados al azar, entonces toma uno de los dos y lo elimina, luego crea un individuo nuevo. Máximo puntaje.
E5	Operador cruce clásico + mutación. Máximo puntaje.
E6	Cada individuo de una población mejora poco a poco. Máximo puntaje.
E7	Mutar dos veces
E1	Operador mutación.
E4	Operador cruce clásico. Toma la parte izquierda del padre A y la parte derecha del padre B. No voltea la cadena.

Tamaño cadena busca: 500 (vocales + consonantes + espacio)

Población: 50 (población pequeña)



E2	Ordena del puntaje mayor a menor la población. Toma el primer individuo, lo copia sobre el último y luego muta esa copia.
E5	Operador cruce clásico + mutación
E3	Operador mutación. Si hay empate en puntaje entre los dos individuos seleccionados al azar, entonces toma uno de los dos y lo elimina, luego crea un individuo nuevo.
E6	Cada individuo de una población mejora poco a poco
E4	Operador cruce clásico. Toma la parte izquierda del padre A y la parte derecha del padre B. No voltea la cadena.
E7	Mutar dos veces
E1	Operador mutación.

Dependiendo del tamaño de la población y de la complejidad de la cadena, algunas estrategias son mejores que otras.

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:

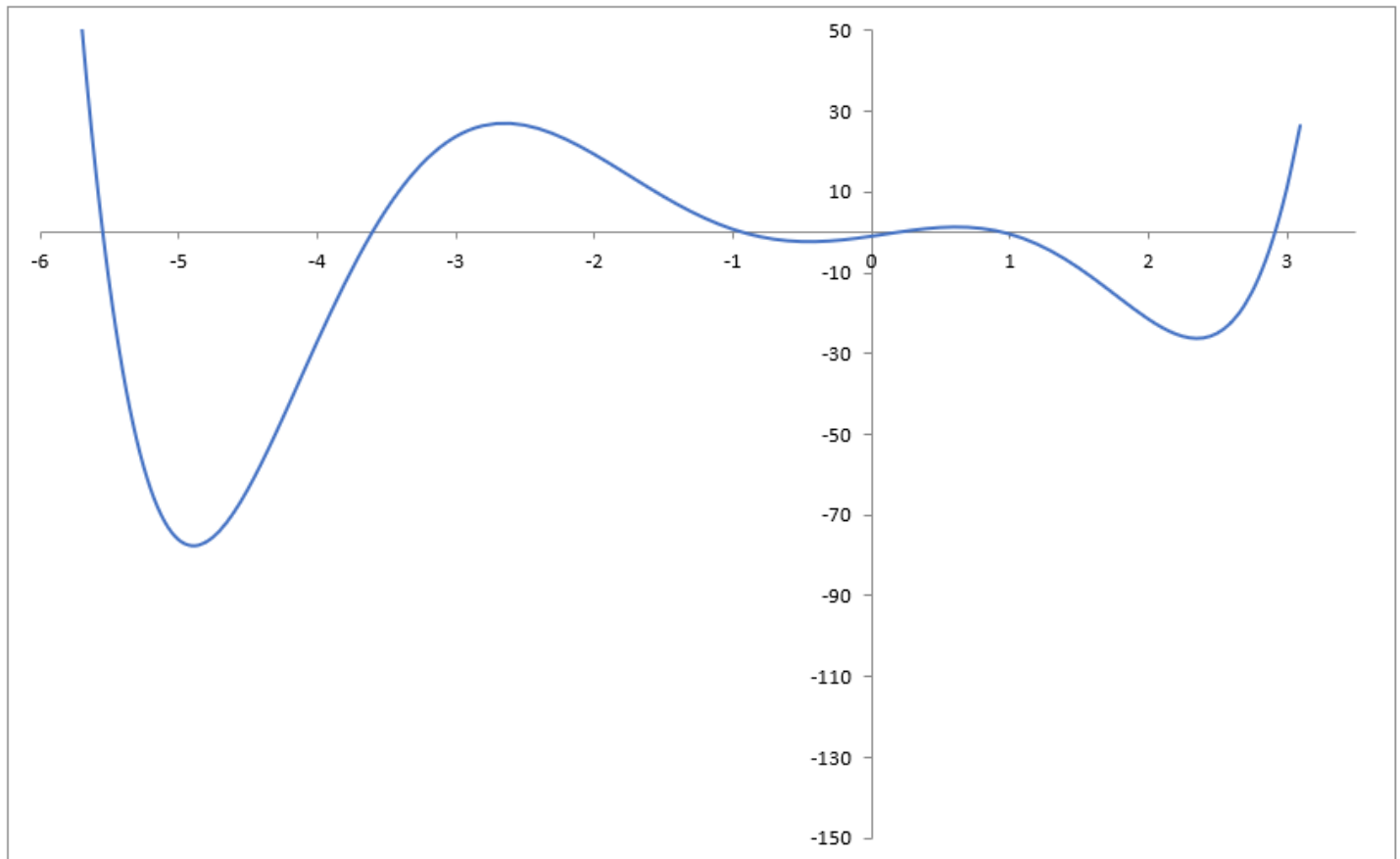


Ilustración 4: Polinomio de grado 6

Para hallar el valor de X que obtenga el mayor valor de Y en el rango X_{min} y X_{max} . 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 X_{min} y X_{max} y para este ejemplo, se usa la distribución uniforme.

En cada ciclo del proceso, se escogen dos individuos al azar y aquel que ofrezca el valor más alto de "Y", es el que se "reproduce" y varía con el operador mutación. ¿Cómo aplicar el operador mutación? Podría ser sumarle o restarle 0.1 o 0.01 o 0.001 o 0.0001 o 0.00001 o 0.000001 y así hasta donde alcance los decimales.

A continuación, el código en C# con el algoritmo:

1. Generar una población de individuos al azar (cada individuo es un número de tipo double entre Xmin y Xmax usando la distribución uniforme).
2. Se escogen dos individuos al azar de esa población.
3. Se califica cada individuo.
4. El mejor individuo, es decir, el que genere un mayor valor de Y es seleccionado y copiado.
5. La copia se modifica ya sea sumando o restándole 0.1 o 0.01 o 0.001 o 0.0001 o 0.00001 o 0.000001.
6. La copia sobrescribe al individuo perdedor.

El código es el siguiente:

J/006.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Buscar el mayor valor de una ecuación
            //modificando números de tipo double
            Poblacion pobl = new();

            int Poblacion = 100;
            int Ciclos = 90000;
            double Xminimo = -4;
            double Xmaximo = 1;
            pobl.Proceso(Poblacion, Ciclos, Xminimo, Xmaximo);
        }
    }

    //Cómo es el individuo
    internal class Individuo {
        public double Genotipo;

        //Al nacer, tendrá un valor double entre 0 y 1
        public Individuo(Random Azar) {
            Genotipo = Azar.NextDouble();
        }

        //Cambia el valor en algún decimal validando que
        //no se salga del rango 0 a 1
        public void Muta(Random Azar) {
            double Copia;
            do {
                Copia = Genotipo;
```

```

    double Divide = Math.Pow(10, Azar.Next(2, 7));
    if (Azar.NextDouble() < 0.5)
        Copia -= 1 / Divide;
    else
        Copia += 1 / Divide;
} while (Copia < 0 || Copia > 1);
Genotipo = Copia;
}
}

//La población
internal class Poblacion {
    public List<Individuo> objInd = [];
    private Random Azar = new();

    public void Proceso(int NumInd, int Ciclos, double Xmin, double Xmax) {
        //Genera la población
        objInd.Clear();
        for (int Contador = 1; Contador <= NumInd; Contador++)
            objInd.Add(new Individuo(Azar));

        //El proceso evolutivo
        for (int Contador = 1; Contador <= Ciclos; Contador++) {
            //Seleccionar al azar dos individuos de esa población: A y B
            int PosA = Azar.Next(objInd.Count);
            int PosB;
            do {
                PosB = Azar.Next(objInd.Count);
            } while (PosB == PosA);

            double ValorXa = objInd[PosA].Genotipo * (Xmax - Xmin) + Xmin;
            double PuntajeA = Ecuacion(ValorXa); //Evaluar adaptación de A

            double ValorXb = objInd[PosB].Genotipo * (Xmax - Xmin) + Xmin;
            double PuntajeB = Ecuacion(ValorXb); //Evaluar adaptación de B

            //Si adaptación de A es mejor que adaptación de B entonces
            if (PuntajeA > PuntajeB) {
                //Eliminar individuo B y duplicar individuo A
                objInd[PosB].Genotipo = objInd[PosA].Genotipo;
                //Modificar levemente al azar el nuevo duplicado
                objInd[PosB].Muta(Azar);
            }
            else {
                //Eliminar individuo A y duplicar individuo B
                objInd[PosA].Genotipo = objInd[PosB].Genotipo;
                //Modificar levemente al azar el nuevo duplicado
                objInd[PosA].Muta(Azar);
            }
        }
    }
}

```



```

    }
}

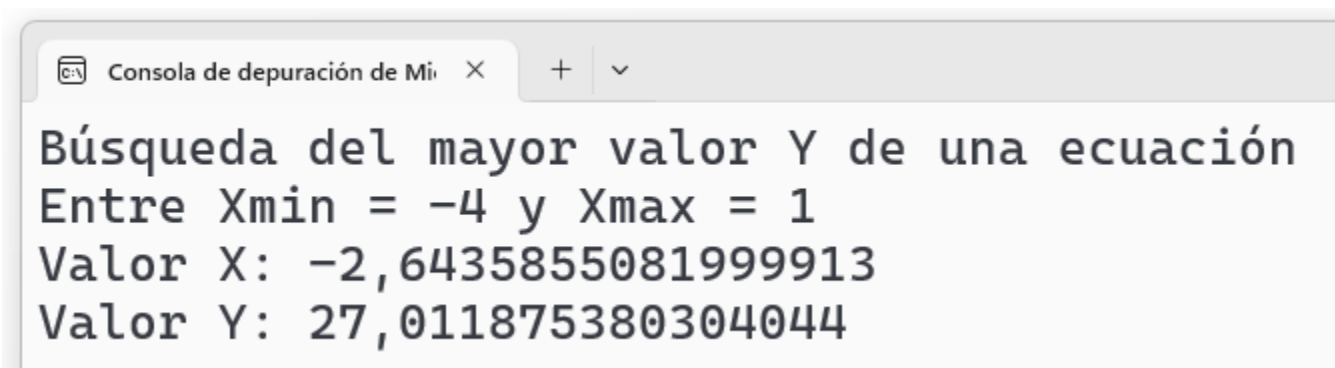
//Buscar individuo con mejor adaptación de la población
double MejorPuntaje = double.MinValue;
int Mejor = 0;
for (int indiv = 0; indiv < objInd.Count; indiv++) {
    double ValorX = objInd[indiv].Genotipo * (Xmax - Xmin) + Xmin;
    double Puntaje = Ecuacion(ValorX);
    if (Puntaje > MejorPuntaje) {
        MejorPuntaje = Puntaje;
        Mejor = indiv;
    }
}

//Imprime el mejor individuo
double MejorValorX = objInd[Mejor].Genotipo * (Xmax - Xmin) + Xmin;

Console.WriteLine("Búsqueda del mayor valor Y de una ecuación");
Console.WriteLine("Entre Xmin = " + Xmin + " y Xmax = " + Xmax);
Console.WriteLine("Valor X: " + MejorValorX);
Console.WriteLine("Valor Y: " + Ecuacion(MejorValorX));
}

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

```



```

Consola de depuración de Mi...
Búsqueda del mayor valor Y de una ecuación
Entre Xmin = -4 y Xmax = 1
Valor X: -2,6435855081999913
Valor Y: 27,011875380304044

```

Ilustración 5: 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 modificar los valores a nivel decimal. ¿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:

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

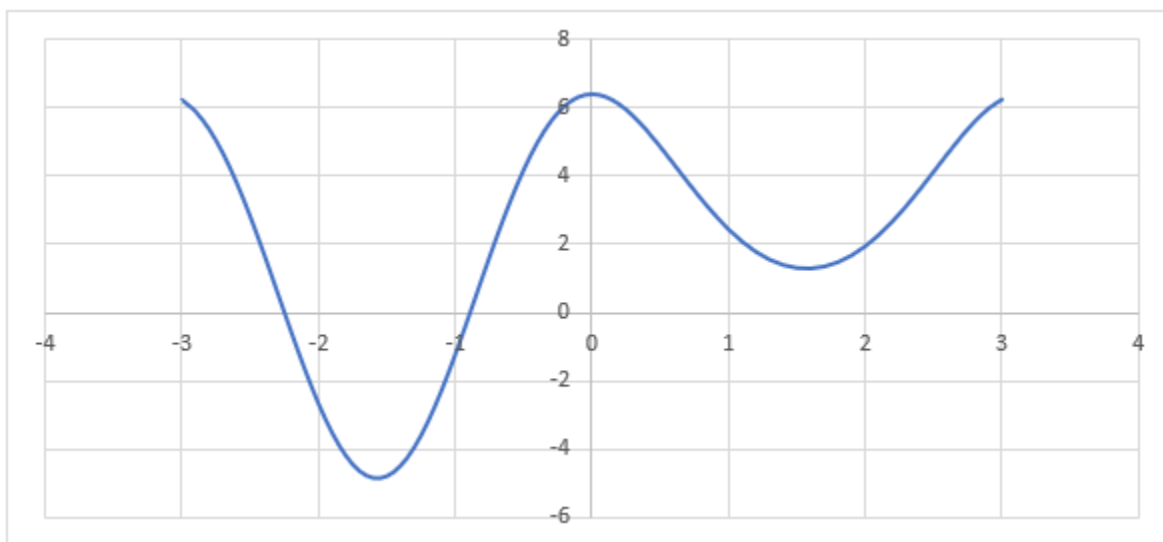


Ilustración 6: Gráfico de una ecuación

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" y el valor máximo 2 será representado por "111111111". ¿Cuántas combinaciones hay? Es $2^{\text{total_cifras}}$, 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:

$$x = \text{RangoMinimo} + \text{valorbinarioadecimal} \times \frac{\text{RangoMáximo} - \text{RangoMínimo}}{2^{\text{TotalCifras}} - 1}$$

Probando con "000000000" (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" (511 en decimal)

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

Probando con "011011101" (221 en decimal)

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

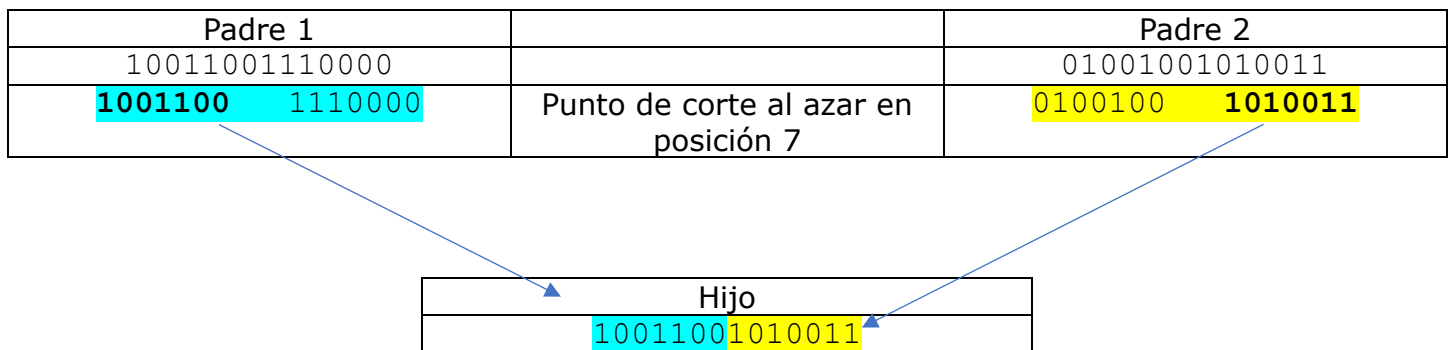
Paso 3

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	10011001 1 10000
Cambiando valor de esa posición	10011001 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:



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 cadenas.

Genotipo: Operador mutación

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
    Evaluar valorY generado por el individuo A
    Evaluar valorY generado por el individuo B
    Si valorY de A es mayor que valorY de B entonces
      Eliminar individuo B
      Duplicar individuo A
      Modificar un bit al azar del nuevo duplicado
    de lo contrario
      Eliminar individuo A
      Duplicar individuo B
      Modificar un bit al azar del nuevo duplicado
    Fin Si
  Fin ciclo
  Buscar individuo que genere mayor Y de la población
  Imprimir individuo
Fin
```

J/007.cs

```
namespace Ejemplo {
  internal class Program {
    static void Main() {
      //Buscar el mayor valor de una ecuación
      //modificando números binarios
      Poblacion pobl = new();

      int NumInd = 100;
      int Ciclos = 90000;
      int Bits = 20;
      double Xmin = -4;
      double Xmax = 1;
      pobl.Proceso(NumInd, Bits, Ciclos, Xmin, Xmax);
    }
  }

  //Cómo es el individuo
  internal class Individuo {
    public int Genotipo;
```

```

//Al nacer, tendrá un valor double entre 0 y 1
public Individuo(Random Azar, int NumeroBits) {
    Genotipo = Azar.Next((int)Math.Pow(2, NumeroBits));
}

//Cambia el valor en algun bit
public void Muta(Random Azar, int NumeroBits) {
    int Mascara = 1 << Azar.Next(NumeroBits);
    Genotipo ^= Mascara;
}
}

//La población
internal class Poblacion {
    public List<Individuo> objInd = [];
    private Random Azar = new();

    public void Proceso(int NumInd, int Bits, int Ciclos,
        double Xmin, double Xmax) {
        //Genera la población
        objInd.Clear();
        for (int Contador = 1; Contador <= NumInd; Contador++)
            objInd.Add(new Individuo(Azar, Bits));

        //El factor de conversión
        double Divide = Math.Pow(2, Bits) - 1;
        double Factor = (Xmax - Xmin) / Divide;

        //El proceso evolutivo
        for (int Contador = 1; Contador <= Ciclos; Contador++) {

            //Seleccionar al azar dos individuos
            //de esa población: A y B
            int PosA = Azar.Next(objInd.Count);
            int PosB;
            do {
                PosB = Azar.Next(objInd.Count);
            } while (PosB == PosA);

            double Xa = Xmin + objInd[PosA].Genotipo * Factor;
            double Pa = Ecuacion(Xa); //Evaluar adaptación de A

            double Xb = Xmin + objInd[PosB].Genotipo * Factor;
            double Pb = Ecuacion(Xb); //Evaluar adaptación de B

            //Si adaptación de A es mejor que adaptación de B entonces
            if (Pa > Pb) {

```

```

        //Eliminar individuo B y duplicar individuo A
        objInd[PosB].Genotipo = objInd[PosA].Genotipo;
        //Modificar levemente al azar el nuevo duplicado
        objInd[PosB].Muta(Azar, Bits);
    }
    else if (Pb > Pa) {
        //Eliminar individuo A y duplicar individuo B
        objInd[PosA].Genotipo = objInd[PosB].Genotipo;
        //Modificar levemente al azar el nuevo duplicado
        objInd[PosA].Muta(Azar, Bits);
    }
}

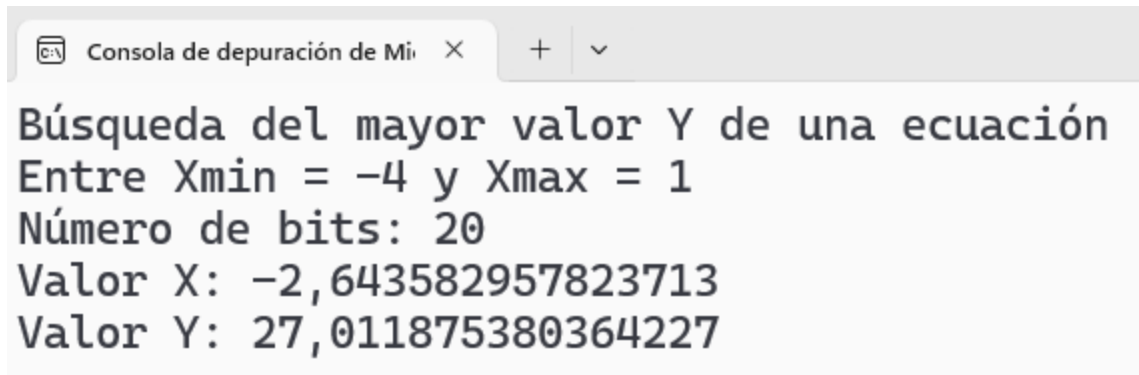
//Buscar individuo con mejor adaptación de la población
double MejorPuntaje = double.MinValue;
int Mejor = 0;
for (int indiv = 0; indiv < objInd.Count; indiv++) {
    double ValorX = Xmin + objInd[indiv].Genotipo * Factor;
    double Puntaje = Ecuacion(ValorX);
    if (Puntaje > MejorPuntaje) {
        MejorPuntaje = Puntaje;
        Mejor = indiv;
    }
}

//Imprime el mejor individuo
double MejorValorX = Xmin + objInd[Mejor].Genotipo * Factor;

Console.WriteLine("Búsqueda del mayor valor Y de una ecuación");
Console.WriteLine("Entre Xmin = " + Xmin + " y Xmax = " + Xmax);
Console.WriteLine("Número de bits: " + Bits);
Console.WriteLine("Valor X: " + MejorValorX);
Console.WriteLine("Valor Y: " + Ecuacion(MejorValorX));
}

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

```



The image shows a screenshot of a debugger's console window. The title bar at the top reads 'Consola de depuración de Mi' followed by a close button 'X'. Below the title bar, the console displays the following text: 'Búsqueda del mayor valor Y de una ecuación', 'Entre Xmin = -4 y Xmax = 1', 'Número de bits: 20', 'Valor X: -2,643582957823713', and 'Valor Y: 27,011875380364227'.

```
Consola de depuración de Mi X + v
Búsqueda del mayor valor Y de una ecuación
Entre Xmin = -4 y Xmax = 1
Número de bits: 20
Valor X: -2,643582957823713
Valor Y: 27,011875380364227
```

Ilustración 7: Búsqueda del mayor valor usando genotipo de cadenas de bits

Genotipo: Operador cruce

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
    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
```

J/008.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Buscar el mayor valor de una ecuación
            //modificando números binarios
            Poblacion pobl = new();

            int NumInd = 1000;
            int Ciclos = 90000;
            int Bits = 20;
            double Xmin = -4;
            double Xmax = 1;
            pobl.Proceso(NumInd, Bits, Ciclos, Xmin, Xmax);
        }
    }

    //Cómo es el individuo
    internal class Individuo {
        public int Genotipo;

        //Al nacer, tendrá un valor dependiendo del número de bits
        public Individuo(Random Azar, int NumeroBits) {
            Genotipo = Azar.Next((int)Math.Pow(2, NumeroBits));
        }

        //Operador cruce.
```

```

public Individuo(Random Azar, int GeneticoA, int GeneticoB) {
    //En que posicion corta el genotipo de cada padre
    int Posicion = Azar.Next(sizeof(int) * 8);

    //Extrae las partes de cada progenitor
    int Mascara = (1 << Posicion) - 1;
    int ParteA = GeneticoA >> Posicion;
    int ParteB = GeneticoB & Mascara;

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

//La población
internal class Poblacion {
    public List<Individuo> objInd = [];
    private Random Azar = new();

    public void Proceso(int NumInd, int Bits, int Ciclos,
        double Xmin, double Xmax) {
        //Genera la población
        objInd.Clear();
        for (int Contador = 1; Contador <= NumInd; Contador++)
            objInd.Add(new Individuo(Azar, Bits));

        //El factor de conversión
        double Divide = Math.Pow(2, Bits) - 1;
        double Factor = (Xmax - Xmin) / Divide;

        //El proceso evolutivo
        for (int Contador = 1; Contador <= Ciclos; Contador++) {
            //Seleccionar al azar dos individuos
            //de esa población: A y B
            int PosA = Azar.Next(objInd.Count);
            int PosB;
            do {
                PosB = Azar.Next(objInd.Count);
            } while (PosB == PosA);

            //Generan un hijo que nace del cruce
            Individuo Hijo = new(Azar, objInd[PosA].Genotipo,
                objInd[PosB].Genotipo);

            double Xa = Xmin + objInd[PosA].Genotipo * Factor;
            double Pa = Ecuacion(Xa); //Evaluar adaptación de A

            double Xb = Xmin + objInd[PosB].Genotipo * Factor;

```

```

double Pb = Ecuacion(Xb); //Evaluar adaptación de B

double ValorXHijo = Xmin + Hijo.Genotipo * Factor;

//Evaluar adaptación de Hijo
double PuntajeHijo = Ecuacion(ValorXHijo);

if (PuntajeHijo > Pa)
    objInd[PosA].Genotipo = Hijo.Genotipo;

if (PuntajeHijo > Pb)
    objInd[PosB].Genotipo = Hijo.Genotipo;
}

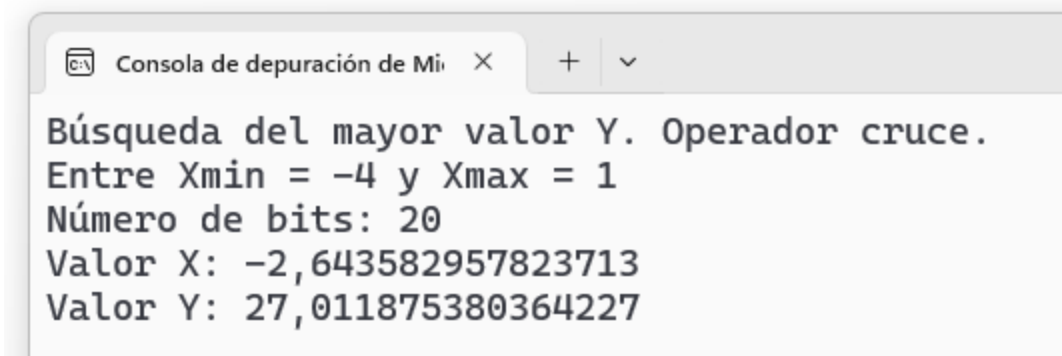
//Buscar individuo con mejor adaptación de la población
double MejorPuntaje = double.MinValue;
int Mejor = 0;
for (int indiv = 0; indiv < objInd.Count; indiv++) {
    double ValorX = Xmin + objInd[indiv].Genotipo * Factor;
    double Puntaje = Ecuacion(ValorX);
    if (Puntaje > MejorPuntaje) {
        MejorPuntaje = Puntaje;
        Mejor = indiv;
    }
}

//Imprime el mejor individuo
double MejorValorX = Xmin + objInd[Mejor].Genotipo * Factor;

Console.WriteLine("Búsqueda del mayor valor Y");
Console.WriteLine(". Operador cruce.");
Console.WriteLine("Entre Xmin = " + Xmin + " y Xmax = " + Xmax);
Console.WriteLine("Número de bits: " + Bits);
Console.WriteLine("Valor X: " + MejorValorX);
Console.WriteLine("Valor Y: " + Ecuacion(MejorValorX));
}

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

```



Consola de depuración de Mi X + v

```
Búsqueda del mayor valor Y. Operador cruce.  
Entre Xmin = -4 y Xmax = 1  
Número de bits: 20  
Valor X: -2,643582957823713  
Valor Y: 27,011875380364227
```

Ilustración 8: Genotipo: Operador cruce

Genotipo: Operador cruce y mutación

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
```

J/009.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Buscar el mayor valor de una ecuación.
            //Operador cruce y mutación
            Poblacion pobl = new();

            int NumInd = 1000;
            int Ciclos = 90000;
            int Bits = 20;
            double Xmin = -4;
            double Xmax = 1;
            pobl.Proceso(NumInd, Bits, Ciclos, Xmin, Xmax);
        }
    }

    //Cómo es el individuo
    internal class Individuo {
        public int Genotipo;

        //Al nacer, tendrá un valor dependiendo del número de bits
        public Individuo(Random Azar, int NumeroBits) {
            Genotipo = Azar.Next((int)Math.Pow(2, NumeroBits));
        }
    }
}
```

```

}

//Operador cruce.
public Individuo(Random Azar, int GeneticoA, int GeneticoB) {
    //En que posicion corta el genotipo de cada padre
    int Posicion = Azar.Next(sizeof(int) * 8);

    //Extrae las partes de cada progenitor
    int Mascara = (1 << Posicion) - 1;
    int ParteA = GeneticoA >> Posicion;
    int ParteB = GeneticoB & Mascara;

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

//Mutación: Cambia el valor en algun bit
public void Muta(Random Azar, int NumeroBits) {
    int Mascara = 1 << Azar.Next(NumeroBits);
    Genotipo ^= Mascara;
}
}

//La población
internal class Poblacion {
    public List<Individuo> objInd = [];
    private Random Azar = new();

    public void Proceso(int NumInd, int Bits, int Ciclos,
        double Xmin, double Xmax) {
        //Genera la población
        objInd.Clear();
        for (int Contador = 1; Contador <= NumInd; Contador++)
            objInd.Add(new Individuo(Azar, Bits));

        //El factor de conversión
        double Divide = Math.Pow(2, Bits) - 1;
        double Factor = (Xmax - Xmin) / Divide;

        //El proceso evolutivo
        for (int Contador = 1; Contador <= Ciclos; Contador++) {

            //Seleccionar al azar dos individuos de esa población: A y B
            int PosA = Azar.Next(objInd.Count);
            int PosB;
            do {
                PosB = Azar.Next(objInd.Count);
            } while (PosB == PosA);
        }
    }
}

```

```

//Generan un hijo que nace del cruce
Individuo Hijo = new(Azar, objInd[PosA].Genotipo,
    objInd[PosB].Genotipo);

//Además muta al Hijo
Hijo.Muta(Azar, Bits);

double Xa = Xmin + objInd[PosA].Genotipo * Factor;
double Pa = Ecuacion(Xa); //Evaluar adaptación de A

double Xb = Xmin + objInd[PosB].Genotipo * Factor;
double Pb = Ecuacion(Xb); //Evaluar adaptación de B

double Xh = Xmin + Hijo.Genotipo * Factor;
double Ph = Ecuacion(Xh); //Evaluar adaptación de Hijo

if (Ph > Pa)
    objInd[PosA].Genotipo = Hijo.Genotipo;

if (Ph > Pb)
    objInd[PosB].Genotipo = Hijo.Genotipo;
}

//Buscar individuo con mejor adaptación de la población
double MejorPuntaje = double.MinValue;
int MejorIndivid = 0;
for (int indiv = 0; indiv < objInd.Count; indiv++) {
    double ValorX = Xmin + objInd[indiv].Genotipo * Factor;
    double Puntaje = Ecuacion(ValorX);
    if (Puntaje > MejorPuntaje) {
        MejorPuntaje = Puntaje;
        MejorIndivid = indiv;
    }
}

//Imprime el mejor individuo
double MejorValorX = Xmin + objInd[MejorIndivid].Genotipo * Factor;

Console.Write("Búsqueda del mayor valor Y");
Console.WriteLine(". Operador cruce y mutación.");
Console.WriteLine("Entre Xmin = " + Xmin + " y Xmax = " + Xmax);
Console.WriteLine("Número de bits: " + Bits);
Console.WriteLine("Valor X: " + MejorValorX);
Console.WriteLine("Valor Y: " + Ecuacion(MejorValorX));
}

public double Ecuacion(double x) {

```

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

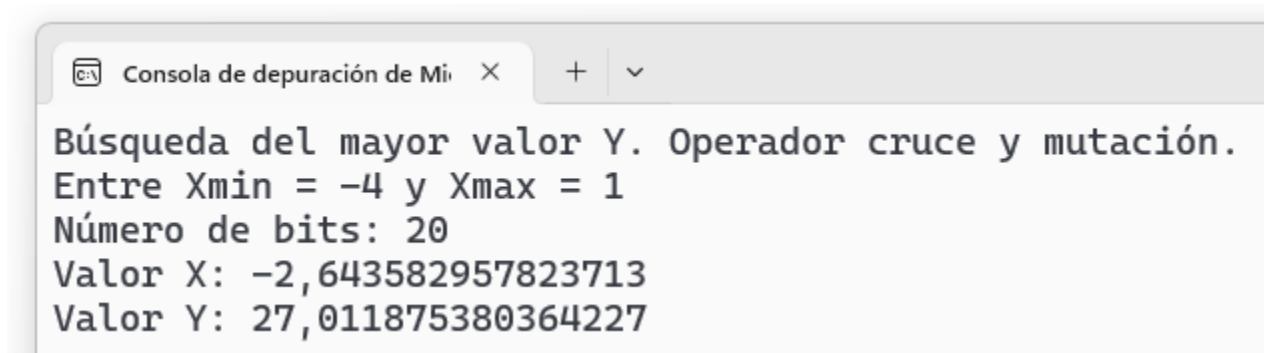


Ilustración 9: Operador cruce y mutación

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.

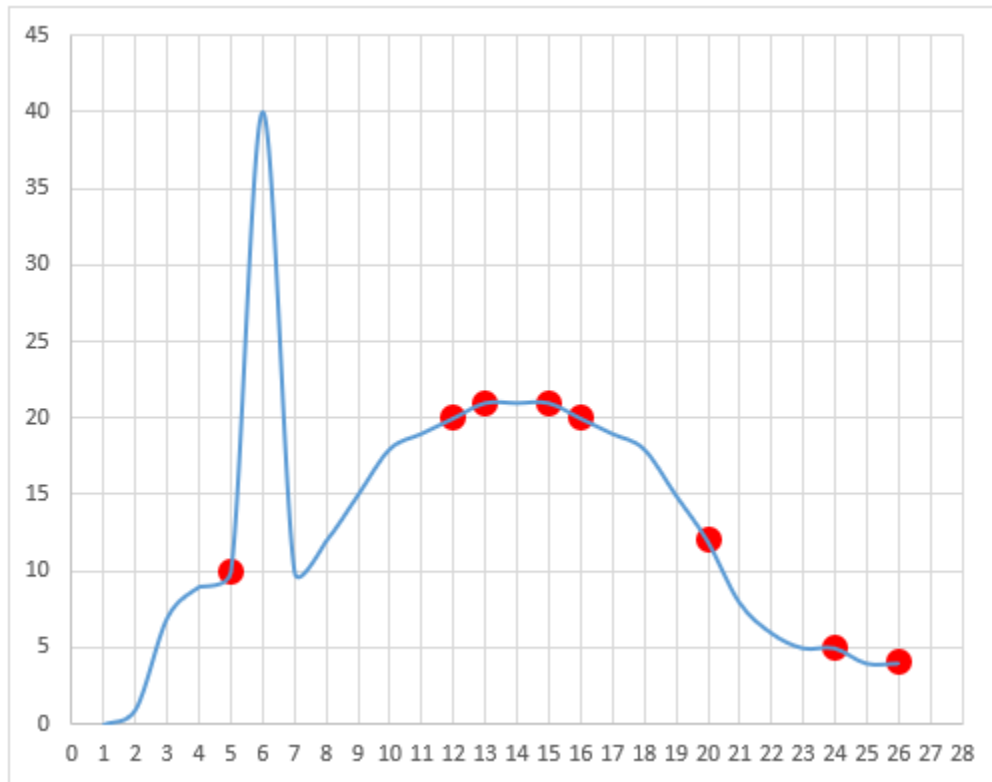


Ilustración 10: Gráfico matemático

Visualmente, el máximo " Y " se encuentra entre los valores de $5 \leq X \leq 7$. Hay muchos individuos entre $11 \leq X \leq 17$ 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 $11 \leq X \leq 17$, y se deduzca erróneamente que el máximo está entre $11 \leq X \leq 17$. 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.

Variando el algoritmo evolutivo

Los algoritmos evolutivos vistos en los ejemplos han sido así: se genera una población de N individuos, se seleccionan dos individuos al azar, se evalúan y comparan sus puntajes, el individuo perdedor es eliminado de la población dejando una vacante, el individuo ganador, en cambio, se premia clonándolo y ese clon se modifica en alguna parte al azar (operador mutación) para posteriormente ocupar la vacante dejada por el individuo eliminado.

Cuando se hace uso del operador cruce, entonces se toman dos individuos al azar, de ellos se genera un hijo al cruzar sus genotipos. Si el hijo resultante, es mejor que alguno de los padres, entonces el o los padres son reemplazados por el hijo en la población.

Tamaño de la población

En los dos casos mencionados anteriormente, la población mantiene una cantidad constante de individuos, pero esto no necesariamente debe ser así. Es posible que un algoritmo evolutivo varíe el tamaño de la población si así lo requiere. Cabe recordar que una población muy grande hará lenta la búsqueda de una solución, pero en una población pequeña se corre el riesgo que se caiga en un mínimo o máximo local, porque se pierde la variedad cuando un individuo es exitoso y su descendencia cubra la población entera.

Selección de individuos

La selección de individuos que compiten entre sí (operador mutación) o para reproducirse (operador cruce) se ha mantenido igual en los diferentes ejemplos. No es necesario limitarse a escoger dos individuos al azar, se puede escoger un número mayor y variar la escogencia del ganador y del perdedor, por ejemplo:

Paso 1: Seleccionar al azar tres individuos de la población

Paso 2: Poner a competir esos tres individuos y seleccionar el mejor

Paso 3: El mejor, se clona y se modifica una parte del clon al azar

Paso 4: Seleccionar al azar tres individuos de la población

Paso 5: Poner a competir esos tres individuos y seleccionar el peor

Paso 6: El peor individuo es reemplazado por el clon mutado generado en el paso 3

Otra forma de seleccionar individuos y que opere el algoritmo evolutivo es así:

Paso 1: Evalúe cada individuo de la población frente al problema y determine su puntaje.

Paso 2: Ordene la población del mejor individuo al peor individuo.

Paso 3: Seleccione un individuo al azar de las primeras posiciones, clónelo, modifíquelo en alguna parte al azar y evalúelo.

Paso 4: Tome un individuo al azar de las últimas posiciones y compare su puntaje con el clon mutado generado en el paso 3. Si el clon mutado es mejor, entonces elimine el individuo y esa vacante es ocupada por el clon mutado. Vuelva al paso 2.

Representación de los individuos

En los ejemplos, el genotipo es de tamaño fijo. La razón es que, dependiendo del problema, el cubrir el rango de soluciones se use un genoma de tamaño fijo. Pero no necesariamente es así, se puede implementar que varíe dinámicamente el tamaño del genoma.

Paralelizar el algoritmo

Los algoritmos evolutivos son candidatos excelentes para ser paralelizados aprovechando que en la actualidad los equipos de cómputo vienen con varios procesadores.

Diversos hilos de ejecución podrían seleccionar individuos de la población y ponerlos a competir entre sí, el único cuidado a tener es que los hilos no seleccionen los mismos individuos. También se podría tener varias poblaciones separadas entre sí y cada hilo aplicaría el algoritmo evolutivo en cada una.

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 Evolutivo

Inicio

Generar población de N individuos (cada uno con valores a, b, c, d, e al azar entre Mínimo y Máximo)

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

Cambiar al azar alguna variable (a, b, c, d, e) del nuevo duplicado de lo contrario

Eliminar individuo A

Duplicar individuo B

Cambiar al azar alguna variable (a, b, c, d, e) del nuevo duplicado

Fin Si

Fin ciclo

Buscar individuo con mejor adaptación de la población

Imprimir individuo

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            //Buscar el mayor valor de una ecuación
            //modificando números de tipo double
            Poblacion pobl = new();

            int NumIndiv = 100;
            int Ciclos = 90000;
            double ValorMinimo = -10;
            double ValorMaximo = 10;
            pobl.Proceso(NumIndiv, Ciclos, ValorMinimo, ValorMaximo);
        }
    }

    //Cómo es el individuo
    internal class Individuo {
        public double valA, valB, valC, valD, valE;

        //Al nacer, tendrá un valor double entre 0 y 1
        public Individuo(Random Azar, double ValMin, double ValMax) {
            valA = Azar.NextDouble() * (ValMax - ValMin) + ValMin;
            valB = Azar.NextDouble() * (ValMax - ValMin) + ValMin;
            valC = Azar.NextDouble() * (ValMax - ValMin) + ValMin;
            valD = Azar.NextDouble() * (ValMax - ValMin) + ValMin;
            valE = Azar.NextDouble() * (ValMax - ValMin) + ValMin;
        }

        //Cambia el valor de una variable
        public void Muta(Random Azar, double ValMin, double ValMax) {
            switch (Azar.Next(5)) {
                case 0:
                    valA = Azar.NextDouble() * (ValMax - ValMin) + ValMin;
                    break;

                case 1:
                    valB = Azar.NextDouble() * (ValMax - ValMin) + ValMin;
                    break;

                case 2:
                    valC = Azar.NextDouble() * (ValMax - ValMin) + ValMin;
                    break;
            }
        }
    }
}
```

```

    case 3:
        valD = Azar.NextDouble() * (ValMax - ValMin) + ValMin;
        break;

    case 4:
        valE = Azar.NextDouble() * (ValMax - ValMin) + ValMin;
        break;
}
}
}

//La población
internal class Poblacion {
    public List<Individuo> objInd = [];
    private Random Azar = new();

    public void Proceso(int NumIndiv, int Ciclos,
        double Minimo, double Maximo) {
        //Genera la población
        objInd.Clear();
        for (int Contador = 1; Contador <= NumIndiv; Contador++)
            objInd.Add(new Individuo(Azar, Minimo, Maximo));

        //El proceso evolutivo
        for (int Contador = 1; Contador <= Ciclos; Contador++) {
            //Seleccionar al azar dos individuos de esa población: A y B
            int PosA = Azar.Next(objInd.Count);
            int PosB;
            do {
                PosB = Azar.Next(objInd.Count);
            } while (PosB == PosA);

            //Evaluar adaptación de A
            double PuntajeA = Ecuacion(objInd[PosA].valA, objInd[PosA].valB,
                objInd[PosA].valC, objInd[PosA].valD,
                objInd[PosA].valE);

            //Evaluar adaptación de B
            double PuntajeB = Ecuacion(objInd[PosB].valA, objInd[PosB].valB,
                objInd[PosB].valC, objInd[PosB].valD,
                objInd[PosB].valE);

            //Si adaptación de A es mejor que adaptación de B entonces
            if (PuntajeA > PuntajeB) {
                //Eliminar individuo B y duplicar individuo A
                objInd[PosB].valA = objInd[PosA].valA;
                objInd[PosB].valB = objInd[PosA].valB;
            }
        }
    }
}

```

```

    objInd[PosB].valC = objInd[PosA].valC;
    objInd[PosB].valD = objInd[PosA].valD;
    objInd[PosB].valE = objInd[PosA].valE;

    //Modificar levemente al azar el nuevo duplicado
    objInd[PosB].Muta(Azar, Minimo, Maximo);
}
else {
    //Eliminar individuo A y duplicar individuo B
    objInd[PosA].valA = objInd[PosB].valA;
    objInd[PosA].valB = objInd[PosB].valB;
    objInd[PosA].valC = objInd[PosB].valC;
    objInd[PosA].valD = objInd[PosB].valD;
    objInd[PosA].valE = objInd[PosB].valE;

    //Modificar levemente al azar el nuevo duplicado
    objInd[PosA].Muta(Azar, Minimo, Maximo);
}
}

//Buscar individuo con mejor adaptación de la población
double MejorPuntaje = double.MinValue;
int Mejor = 0;
for (int indiv = 0; indiv < objInd.Count; indiv++) {
    double Puntaje = Ecuacion(objInd[indiv].valA, objInd[indiv].valB,
        objInd[indiv].valC, objInd[indiv].valD,
        objInd[indiv].valE); ;
    if (Puntaje > MejorPuntaje) {
        MejorPuntaje = Puntaje;
        Mejor = indiv;
    }
}

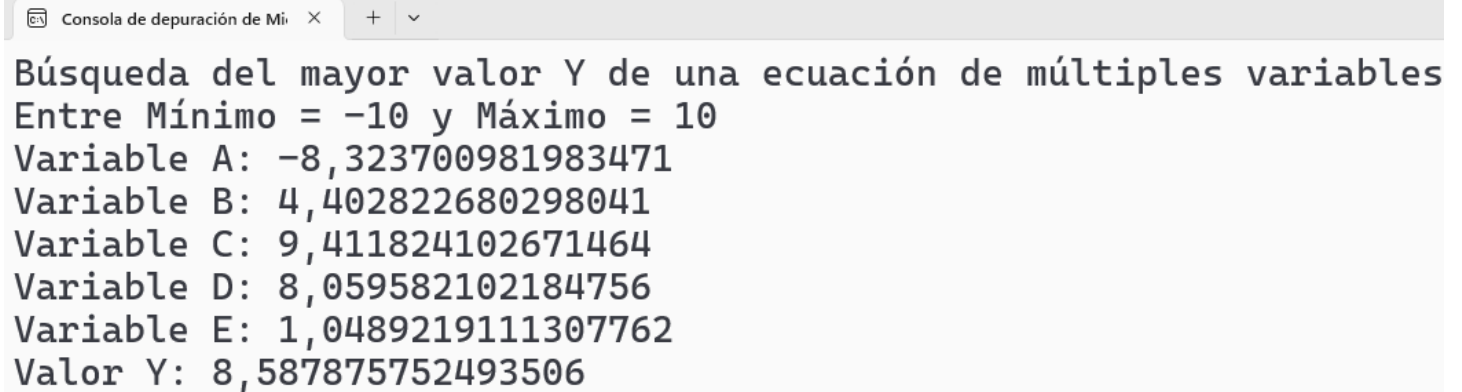
//Imprime el mejor individuo
Console.Write("Búsqueda del mayor valor Y");
Console.WriteLine(" de una ecuación de múltiples variables");
Console.Write("Entre Mínimo = " + Minimo);
Console.WriteLine(" y Máximo = " + Maximo);
Console.WriteLine("Variable A: " + objInd[Mejor].valA);
Console.WriteLine("Variable B: " + objInd[Mejor].valB);
Console.WriteLine("Variable C: " + objInd[Mejor].valC);
Console.WriteLine("Variable D: " + objInd[Mejor].valD);
Console.WriteLine("Variable E: " + objInd[Mejor].valE);
Console.WriteLine("Valor Y: " + MejorPuntaje);
}

public double Ecuacion(double a, double b, double c,
    double d, double e) {

```



```
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);  
}  
}  
}
```



Consola de depuración de Mi + ▼

Búsqueda del mayor valor Y de una ecuación de múltiples variables
Entre Mínimo = -10 y Máximo = 10
Variable A: -8,323700981983471
Variable B: 4,402822680298041
Variable C: 9,411824102671464
Variable D: 8,059582102184756
Variable E: 1,0489219111307762
Valor Y: 8,587875752493506

Ilustración 11: Buscar el mayor valor de Y con múltiples variables independientes

Simplificar una ecuación

Dada una ecuación original del tipo:

$$Y = a * \text{seno}(b * X + c) + d * \text{seno}(e * X + f) + \dots + p * \text{seno}(q * X + r)$$

Como se puede observar, tiene varios bloques del tipo:

$$Y = a * \text{seno}(b * X + c)$$

¿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 6 bloques que se acerque lo más posible a lo que hace la de 30.

J/011.cs

```
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 {

    class Program {
        static void Main() {
            Random Azar = new(); //Generador de números aleatorios único
```

```

//=====
//Configuración básica del simulador
//=====

//Cuántos individuos tendrá la población
int NumIndiv = 200;

//Cuántos milisegundos dará al
//algoritmo evolutivo para operar
long Tiempo = 20000;

//Cuántos datos generará la ecuación compleja
int TotalDatos = 100;

//Valor mínimo de la variable X
//para el conjunto de datos
double Xmin = -720;

//Valor máximo de la variable X
//para el conjunto de datos
double Xmax = 720;

//=====
//Configuración avanzada del simulador
//=====

//Y = a*seno(b*x+c) + d*seno(e*x+f) + ... +
//coef0*seno(coef1*x+coef2) + coef3*seno(coef4*x+coef5)

//Cada coef0*seno(coef1*x+coef2) se le conoce como bloque

//Número de bloques de la ecuación que genera el dataset
int BloquesDataset = 30;

//Número de bloques de la ecuación del individuo
int BloqueIndiv = 6;

//Valor mínimo que tendrán los coeficientes
double CfMin = -3;

//Valor máximo que tendrán los coeficientes
double CfMax = 3;

//=====
//Imprime los datos de la simulación
//=====
Console.WriteLine("Algoritmo Evolutivo\r\n");

```

```

Console.WriteLine("Individuos por población: " + NumIndiv);
Console.WriteLine("Milisegundos: " + Tiempo);
Console.WriteLine("Datos generados: " + TotalDatos);
Console.WriteLine("Rango de datos en X entre: " + Xmin + " y " + Xmax);
Console.WriteLine("Ecuación compleja. Bloques: " + BloquesDataset);
Console.WriteLine("Ecuación simple. Bloques: " + BloqueIndiv);
Console.Write("Coeficientes entre: " + CfMin);
Console.WriteLine(" y " + CfMax);

//Prepara el generador de datos
GeneraEcuacion Genera = new();

//Crea un dataset de valores X, Y
Genera.GeneraDatos(Azar, BloquesDataset, CfMin, CfMax,
    Xmin, Xmax, TotalDatos);

//Configura la población que se
//adaptará al dataset
Poblacion poblacion = new(Azar, NumIndiv, BloqueIndiv, CfMin, CfMax);

//Proceso de buscar el mejor individuo
//al dataset generado
poblacion.Proceso(Azar, Genera.Xentrada,
    Genera.Yesperado, Tiempo);
}
}

//Crea una ecuación compleja al azar
public class GeneraEcuacion {
    //Coeficientes de la ecuación que representa a los datos del dataset
    //Y = a*seno(b*x+c) + d*seno(e*x+f) + ... +
    // coef0*seno(coef1*x+coef2) + coef3*seno(coef4*x+coef5)
    //
    //Cada coef0*seno(coef1*x+coef2) se le conoce como bloque
    List<double>? Cf;

    //Valores X de entrada
    public List<double> Xentrada = [];

    //Valores Y generados por la ecuación
    public List<double> Yesperado = [];

    //Genera los datos del dataset
    public void GeneraDatos(Random Azar, int NumBloques,
        double CfMin, double CfMax,
        double Xmin, double Xmax, int TotalDatos) {
        //Coeficientes al azar
        Cf = [];
    }
}

```

```

for (int Cont = 1; Cont <= NumBloques * 3; Cont++)
    Cf.Add(Azar.NextDouble() * (CfMax - CfMin) + CfMin);

Xentrada.Clear();
Yesperado.Clear();

//Valores de X que tendrá.
for (int Cont = 1; Cont <= TotalDatos; Cont++)
    Xentrada.Add(Azar.NextDouble() * (Xmax - Xmin) + Xmin);
Xentrada.Sort();

//Genera el dataset con esta ecuación compleja
for (int Xval = 0; Xval < Xentrada.Count; Xval++) {
    double Y = 0;
    for (int Cont = 0; Cont < Cf.Count; Cont += 3) {
        double Valor = Cf[Cont + 1] * Xentrada[Xval] + Cf[Cont + 2];
        Y += Cf[Cont] * Math.Sin(Valor * Math.PI / 180);
    }
    Yesperado.Add(Y);
}
}
}

internal class Poblacion {
    //Almacena los individuos de la población
    List<Individuo> Individuos = [];

    //Inicializa la población con los individuos
    public Poblacion(Random Azar, int numIndividuos,
        int BloqInd,
        double CfMin, double CfMax) {
        Individuos.Clear();
        for (int cont = 1; cont <= numIndividuos; cont++)
            Individuos.Add(new Individuo(Azar, BloqInd, CfMin, CfMax));
    }

    //Proceso evolutivo.
    public void Proceso(Random Azar, List<double> Xentra,
        List<double> Yespera, long Tiempo) {

        //Medidor de tiempos
        Stopwatch cronometro = new();
        cronometro.Reset();
        cronometro.Start();

        //Tiempo que repetirá el proceso evolutivo
        while (cronometro.ElapsedMilliseconds < Tiempo) {

```

```

//Escoge dos individuos al azar
int indivA = Azar.Next(Individuos.Count);
int indivB;
do {
    indivB = Azar.Next(Individuos.Count);
} while (indivB == indivA);

//Evalúa cada individuo con respecto a las
//entradas y salidas esperadas
Individuos[indivA].AjusteIndiv(Xentra, Yespera);
Individuos[indivB].AjusteIndiv(Xentra, Yespera);

//El mejor individuo genera una copia
//que sobrescribe al peor y la copia se muta
if (Individuos[indivA].Ajuste < Individuos[indivB].Ajuste)
    CopiaMuta(Azar, indivA, indivB);
else
    CopiaMuta(Azar, indivB, indivA);
}

//Imprime el mejor individuo
int MejorIndividuo = -1;
double MejorAjuste = double.MaxValue;
for (int cont = 0; cont < Individuos.Count; cont++) {

    if (Individuos[cont].Ajuste != -1 &&
        Individuos[cont].Ajuste < MejorAjuste) {
        MejorAjuste = Individuos[cont].Ajuste;
        MejorIndividuo = cont;
    }

}

Individuos[MejorIndividuo].Muestra(Xentra, Yespera);
}

public void CopiaMuta(Random Azar, int Origen, int Destino) {
    Individuo ganador, perdedor;
    ganador = Individuos[Origen];
    perdedor = Individuos[Destino];

    //Copia el individuo
    for (int Cont = 0; Cont < ganador.Coeff.Count; Cont++)
        perdedor.Coeff[Cont] = ganador.Coeff[Cont];

    //Muta la copia
    perdedor.Muta(Azar);
}

```

```

}

internal class Individuo {
    //Coeficientes de la ecuación que representa al individuo
    //Y = a*seno(b*x+c) + d*seno(e*x+f) + ... +
    //coef0*seno(coef1*x+coef2) + coef3*seno(coef4*x+coef5)

    //Cada coef0*seno(coef1*x+coef2) se le conoce como bloque
    public List<double>? Coef;

    //Guarda en "caché" el ajuste para no tener
    //que calcularlo continuamente
    public double Ajuste;

    //Inicializa el individuo con las Piezas,
    //Modificadores y Operadores al azar
    public Individuo(Random Azar, int BloqInd,
        double CfMin, double CfMax) {
        //Coeficientes al azar
        Coef = [];
        for (int Cont = 1; Cont <= BloqInd * 3; Cont++)
            Coef.Add(Azar.NextDouble() * (CfMax - CfMin) + CfMin);
        Ajuste = -1;
    }

    //Calcula el ajuste del individuo con los valores de salida esperados
    public void AjusteIndiv(List<double> Entradas, List<double> Yesperado) {
        //Si ya había sido calculado entonces
        //evita calcularlo de nuevo
        if (Ajuste != -1) return;

        //Deduce el ajuste
        Ajuste = 0;
        for (int Xval = 0; Xval < Entradas.Count; Xval++) {
            double Y = 0;
            for (int Cont = 0; Cont < Coef.Count; Cont += 3) {
                double Valor = Coef[Cont + 1] * Entradas[Xval] + Coef[Cont + 2];
                Y += Coef[Cont] * Math.Sin(Valor * Math.PI / 180);
            }

            //Diferencia entre la salida calculada y la esperada
            Ajuste += Math.Abs(Y - Yesperado[Xval]);
        }
    }

    //Muta alguna parte del individuo
    public void Muta(Random Azar) {
        int CualMuta = Azar.Next(Coef.Count);
    }
}

```

```

    Coef[CualMuta] += Azar.NextDouble() * 2 - 1;
    Ajuste = -1;
}

//Imprime el individuo frente a los datos esperados
public void Imprime(List<double> Entradas, List<double> Yesperado) {
    Console.WriteLine("\r\n\r\nIndividuo");
    Console.WriteLine("Ajuste: " + Ajuste);
    for (int Cont = 0; Cont < Coef.Count; Cont++)
        Console.WriteLine("Coef: " + Coef[Cont]);

    Console.WriteLine("\r\nEntrada;Salida Esperada;Salida Calculada");
    for (int Xval = 0; Xval < Entradas.Count; Xval++) {
        double Y = 0;
        for (int Cont = 0; Cont < Coef.Count; Cont += 3) {
            double Valor = Coef[Cont + 1] * Entradas[Xval] + Coef[Cont + 2];
            Y += Coef[Cont] * Math.Sin(Valor * Math.PI / 180);
        }
        Console.Write(Entradas[Xval] + ";" + Yesperado[Xval]);
        Console.WriteLine("; " + Y);
    }
}

//Imprime el individuo con los datos normalizados
public void Muestra(List<double> Entradas,
    List<double> Yesperado) {

    Console.WriteLine("\r\n\r\nIndividuo. Normalizado.");

    double Xmin = double.MaxValue;
    double Ymin = double.MaxValue;
    double Xmax = double.MinValue;
    double Ymax = double.MinValue;

    //Máximos y mínimos del dataset
    for (int Xval = 0; Xval < Entradas.Count; Xval++) {

        if (Entradas[Xval] < Xmin)
            Xmin = Entradas[Xval];

        if (Entradas[Xval] > Xmax)
            Xmax = Entradas[Xval];

        if (Yesperado[Xval] < Ymin)
            Ymin = Yesperado[Xval];

        if (Yesperado[Xval] > Ymax)
            Ymax = Yesperado[Xval];
    }
}

```



```

}

//Salidas del individuo
List<double> IndivSale = [];
for (int Xval = 0; Xval < Entradas.Count; Xval++) {
    double Y = 0;
    for (int Cont = 0; Cont < Coef.Count; Cont += 3) {
        double Valor = Coef[Cont + 1] * Entradas[Xval] + Coef[Cont + 2];
        Y += Coef[Cont] * Math.Sin(Valor * Math.PI / 180);
    }
    IndivSale.Add(Y);
    if (Y < Ymin) Ymin = Y;
    if (Y > Ymax) Ymax = Y;
}

//Normaliza
double AjusteNorm = 0;
for (int Xval = 0; Xval < Entradas.Count; Xval++) {
    Entradas[Xval] = (Entradas[Xval] - Xmin) / (Xmax - Xmin);
    Yesperado[Xval] = (Yesperado[Xval] - Ymin) / (Ymax - Ymin);
    IndivSale[Xval] = (IndivSale[Xval] - Ymin) / (Ymax - Ymin);
    AjusteNorm += Math.Abs(Yesperado[Xval] - IndivSale[Xval]);
}

//Imprime
Console.WriteLine("Ajuste: " + AjusteNorm);
Console.WriteLine("\r\nEntrada;Salida Esperada;Salida Calculada");
for (int Xval = 0; Xval < Entradas.Count; Xval++) {
    Console.Write(Entradas[Xval] + ";" + Yesperado[Xval]);
    Console.WriteLine("; " + IndivSale[Xval]);
}
}
}
}
}

```

Ejemplo de ejecución:

Algoritmo Evolutivo

Individuos por población: 200

Milisegundos: 20000

Datos generados: 100

Rango de datos en X entre: -720 y 720

Ecuación compleja. Bloques: 30

Ecuación simple. Bloques: 6

Coeficientes entre: -3 y 3

Individuo. Normalizado.

Ajuste: 3,041673065979144

Entrada;Salida Esperada;Salida Calculada

0;0,19076488073466807;0,19075304053200304

0,009196495664768083;0,19056428770740783;0,20692920868395803

0,017183593777794513;0,2606757850350473;0,2750747777308636

0,024240778867472737;0,36399303131902955;0,36572987453823985

0,025366125391829704;0,3826419276398382;0,3817385984692082

0,0396795519187209;0,6122109100658066;0,5779682171515831

0,04646873004921072;0,6802365859976393;0,6392448383087276

0,05835168688957167;0,686131215422329;0,6577802064286983

0,06806822254492467;0,5914361617222719;0,5912792108986039

0,06839262548504865;0,5872015312323271;0,588101970235629

Ilustración 12: Algoritmo Evolutivo

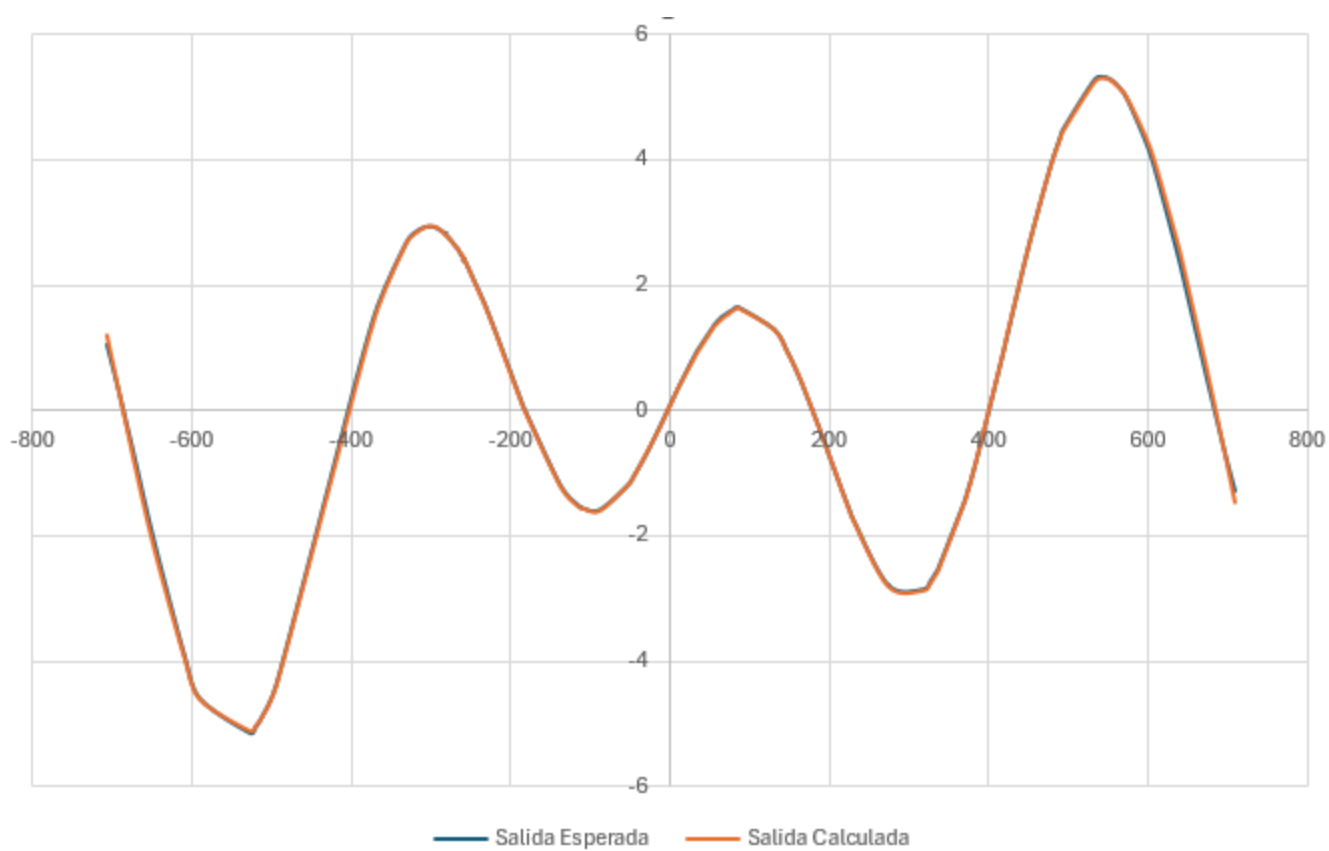


Ilustración 13: Ejemplo de simplificación de curvas

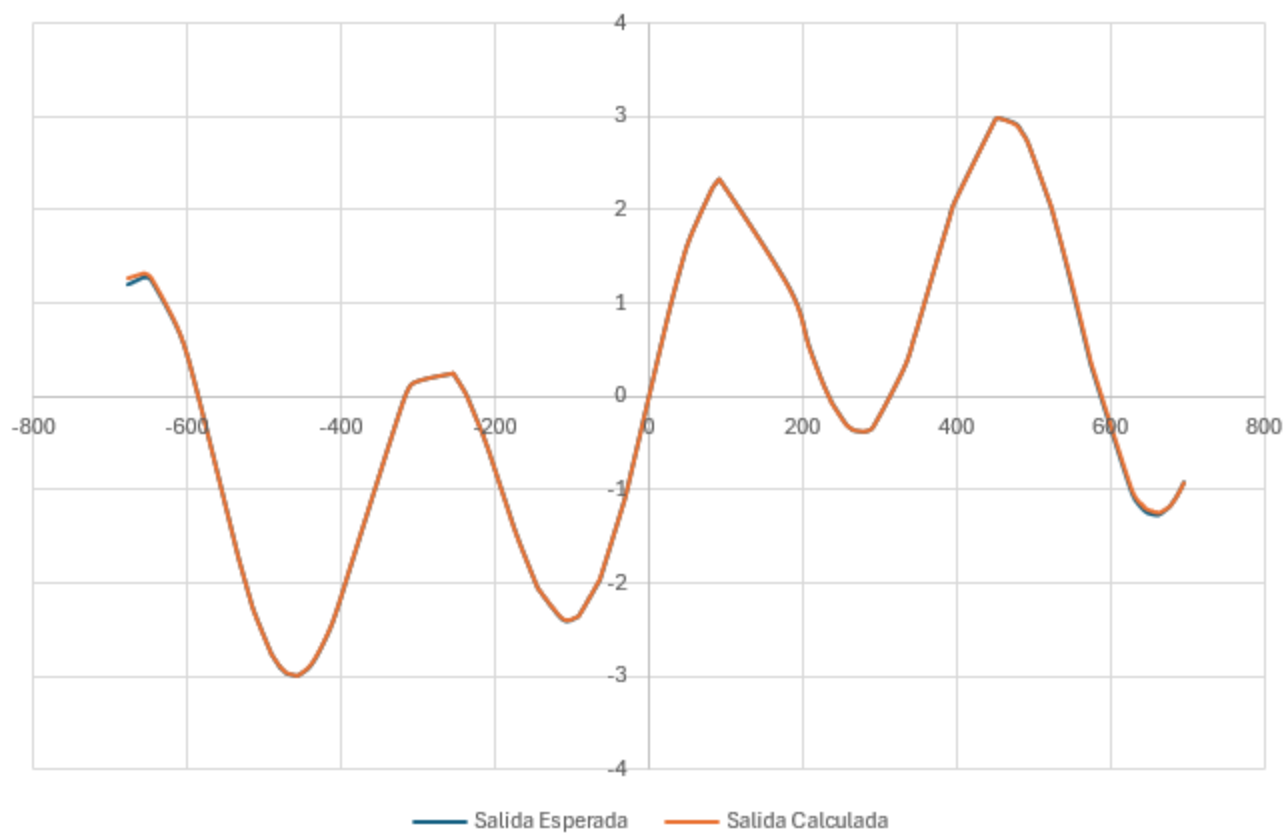


Ilustración 14: Ejemplo de simplificación de curvas