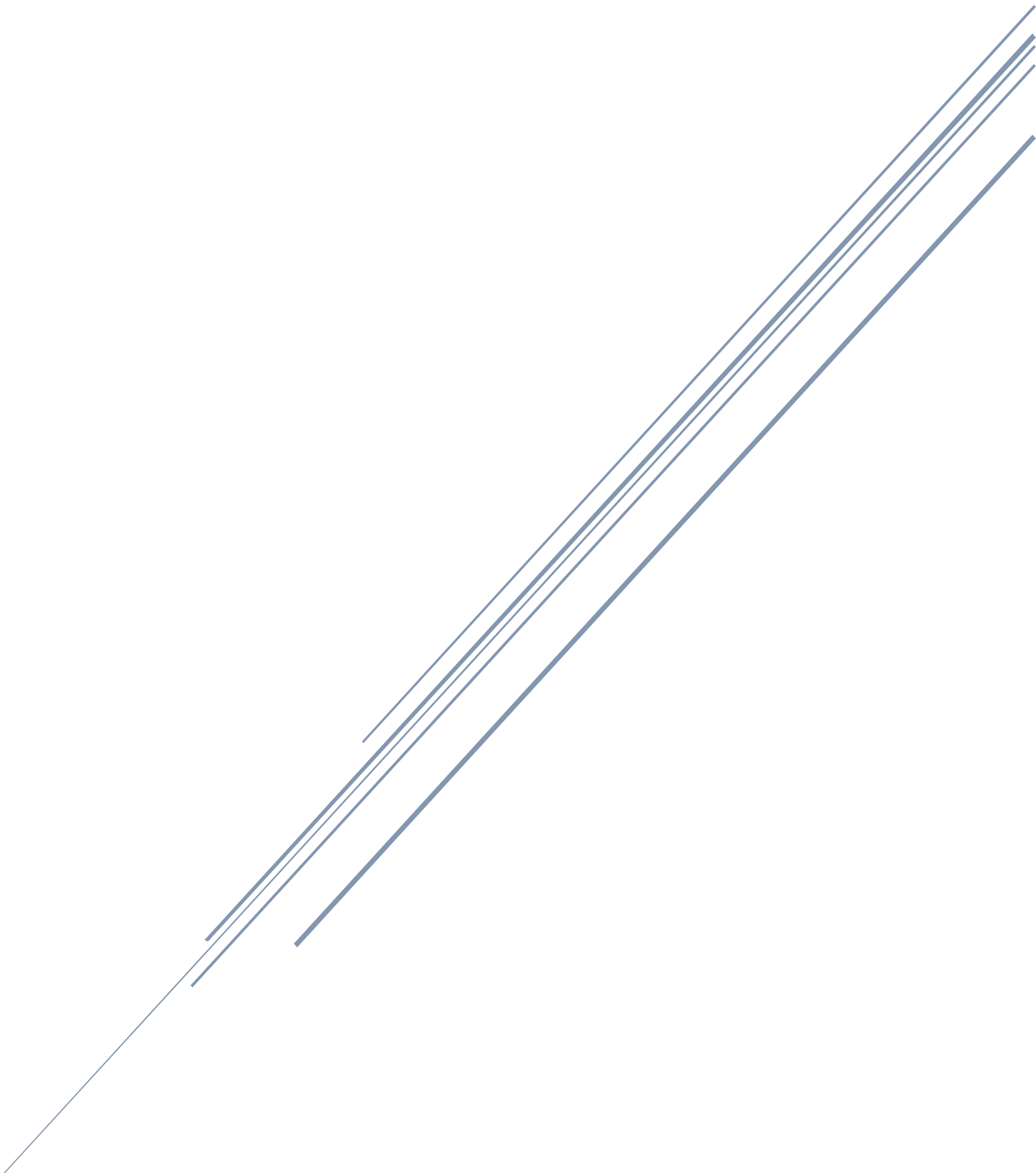


ALGORITMOS GENÉTICOS (2020)

Implementación de algoritmos genéticos para resolver diversos problemas

Autor: Rafael Alberto Moreno Parra



Contenido

Licencia del software 2

Marcas registradas 2

Introducción..... 3

Sobre los generadores de números aleatorios 4

Cuidados a tener al usar los generadores 6

Iniciando con algoritmos genéticos..... 8

 El problema 8

 Algoritmo Genético: el operador Mutación 10

 Algoritmo Genético: el operador Cruce..... 14

 Algoritmo Genético: Combinando los operadores cruce y mutación 17

 Consideraciones 20

Buscar el máximo en una función 21

De genotipos y fenotipos 24

 Operador mutación 26

 Operador cruce 30

 Combinando mutación y cruce 33

Trazando el camino 36

 Una consideración sobre este algoritmo 40

Juego de aritmética 41

El problema de las Reinas..... 46

Creando Sudokus..... 51

 Factor de evaluación. Clave para los algoritmos genéticos. 51

 Peor caso 51

 Mejor caso..... 52

Máximos y mínimos locales..... 56

Variando el algoritmo genético 57

 Algoritmo usado en este libro 57

 Tamaño de la población..... 57

 Selección de individuos 57

 Representación de los individuos..... 57

 Paralelizar el algoritmo 57

Conclusiones 58

Bibliografía..... 59

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 2019 ® Enlace: <http://windows.microsoft.com/en-US/windows/home>

Introducción

Los algoritmos genéticos son una rama de la inteligencia artificial que se inspiran en los seres vivos y más precisamente en la Teoría de la Evolución de las Especies. A grandes rasgos, consiste que dado un problema expresado como un ambiente donde habitan diversos individuos. Estos individuos comienzan a “vivir” en ese ambiente y sólo aquellos más aptos pueden prosperar. Esa prosperidad se expresa en tener mayor descendencia. Los hijos heredan los caracteres de su único progenitor (reproducción asexual) o sus progenitores (reproducción sexual) y los varían sea por la combinación de los genes de sus progenitores (en el caso de reproducción sexual) o por mutación o por ambos. Esas variaciones pueden mejorar la adaptación o empeorarla, es aleatorio, para aquellos que mejoren su adaptación el premio es dejar mayor descendencia. Con el pasar de las generaciones, la población tendrá individuos cada vez más adaptados. Esos individuos más adaptados son la solución al problema.

Nota: Todo el código fuente de este libro puede encontrarlo en:

<https://github.com/ramsoftware/LibroAlgoritmoGenetico2020>

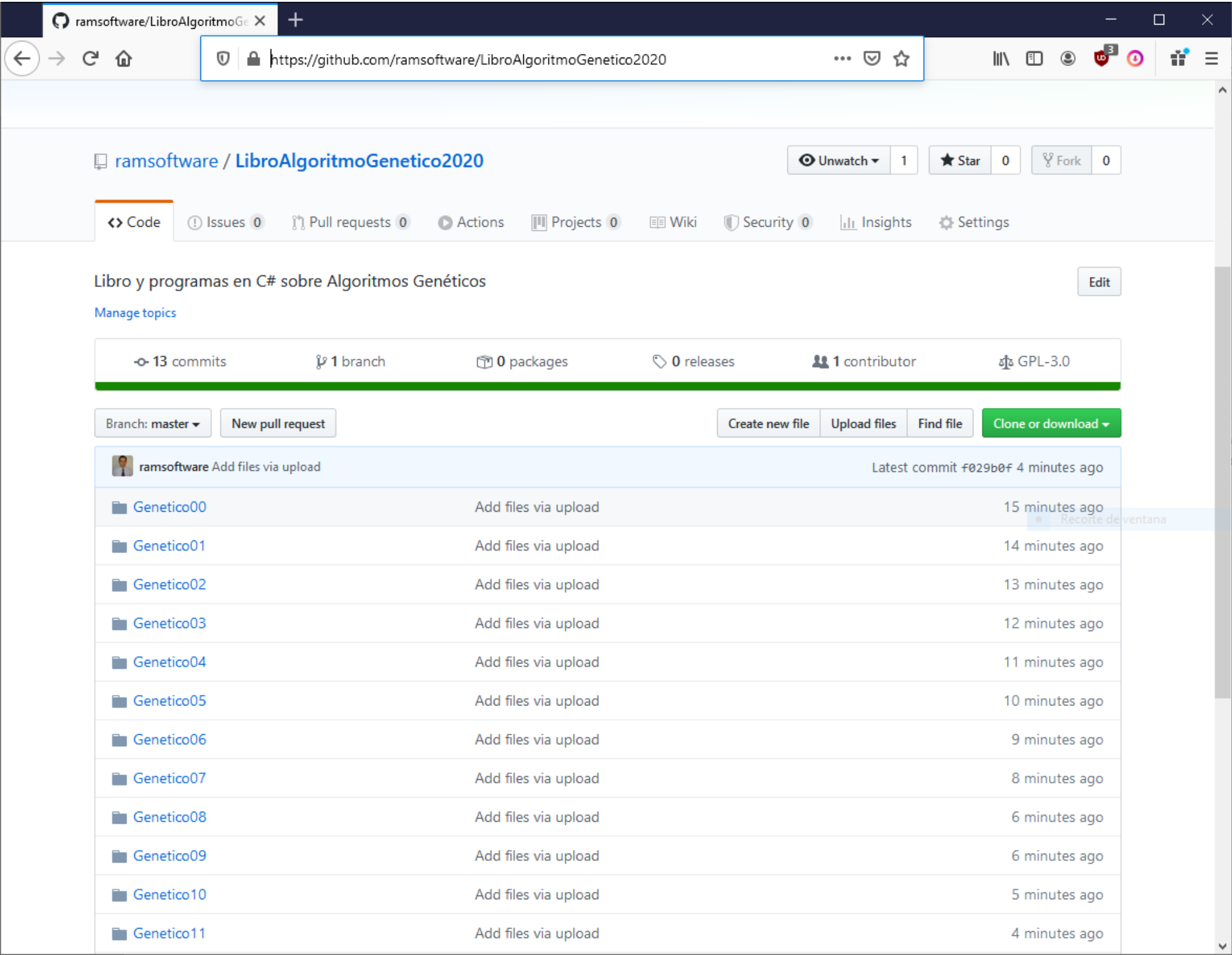


Ilustración 1: <https://github.com/ramsoftware/LibroAlgoritmoGenetico2020>

Sobre los generadores de números aleatorios

Es intensivo el uso de números aleatorios al trabajar con algoritmos genéticos. El azar puro es difícil de conseguir (hay que recurrir a hardware especial o servicios en Internet como <https://www.random.org/>), por lo tanto, se trabaja con generadores de números pseudo-aleatorios, que no son azar puro, pero cumplen con la premisa de ser confiables, es decir, que pasan las pruebas de independencia y uniformidad. En este libro se hará uso de generadores de números pseudo-aleatorios.

Es necesario investigar que generador usa el lenguaje de programación que se vaya a usar. En el caso de Random de .NET, este se explica aquí: <https://docs.microsoft.com/en-us/dotnet/api/system.random?view=netframework-4.8> . Otros aspectos importantes que se debe mirar con cuidado es el período (cantidad de números aleatorios que se generan antes que se empiece a repetir la serie) y su velocidad de generación (por supuesto, debe ser muy rápido). Un generador como Mersenne Twister en <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html> presume de un período gigantesco y es más rápido que un generador congruencial lineal que es de uso común en los lenguajes de programación. Ver: <https://stackoverflow.com/questions/32027839/why-is-mersenne-twister-faster-than-the-linear-congruential-generator> .

Para saber más, este es el código fuente de la clase Random de .NET:

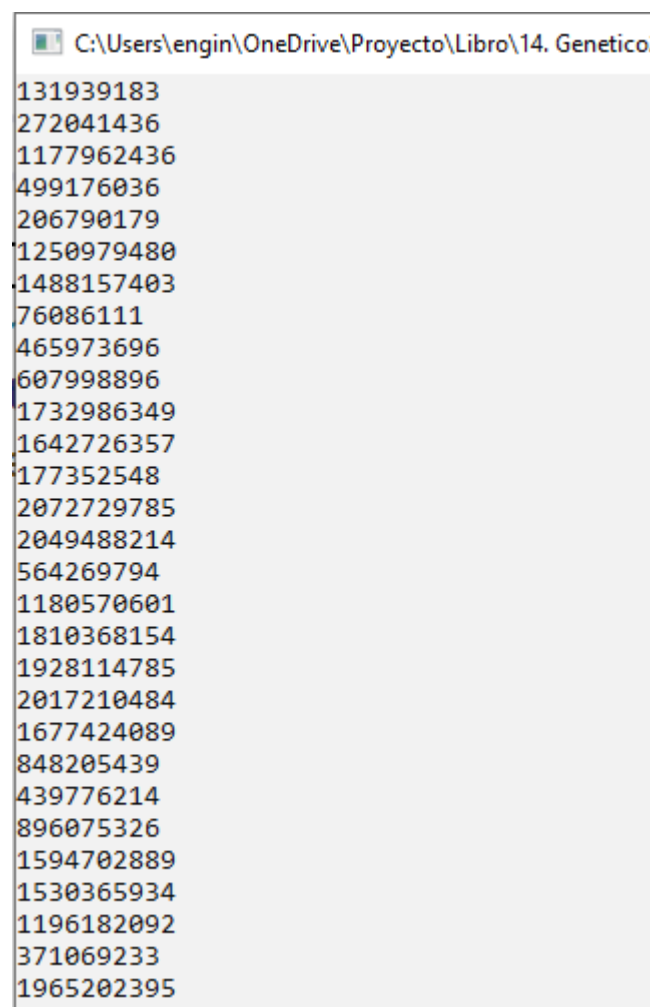
<https://referencesource.microsoft.com/#mscorlib/system/random.cs>

Random de .NET no es un generador congruencial lineal sino un generador sustractivo: <https://csharp.dokry.com/1738/como-se-genera-un-numero-aleatorio-en-tiempo-de-ejecucion.html> que está bien para nuestro trabajo con algoritmos genéticos pero **no** se debe usar en criptografía: <https://codingvision.net/security/c-predict-random-number-generator-net>

En C# así sería un ejemplo de imprimir números pseudo-aleatorios enteros positivos incluyendo el cero:

```
using System;

namespace Aleatorio {
    class Program {
        static void Main(string[] args) {
            Random azar = new Random();
            for (int cont=1; cont<=40; cont++) {
                Console.WriteLine(azar.Next());
            }
            Console.ReadKey();
        }
    }
}
```



C:\Users\engin\OneDrive\Proyecto\Libro\14. Genetico

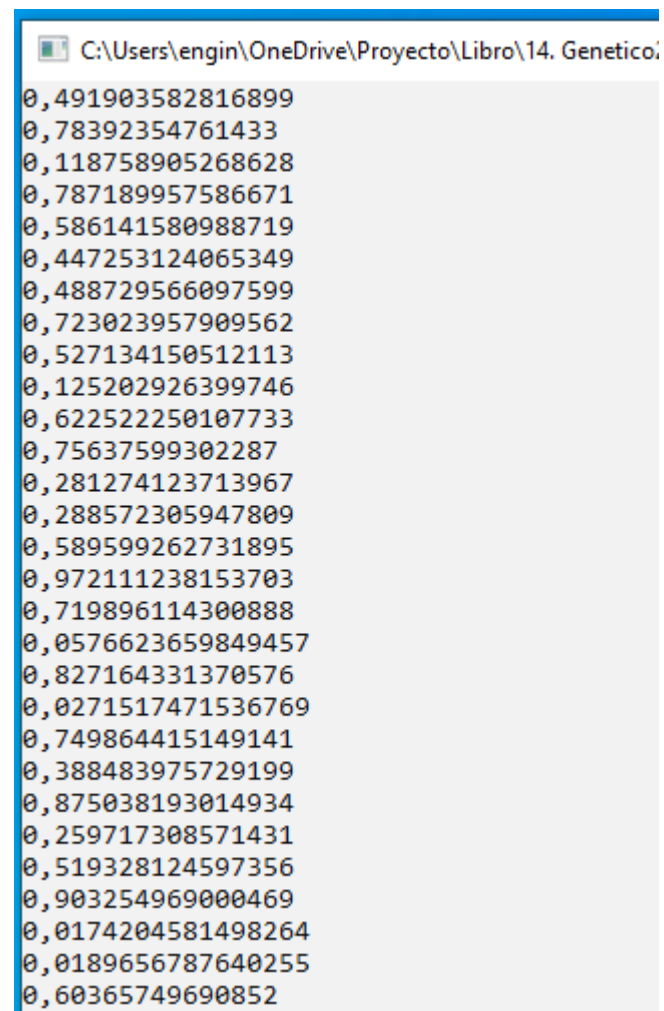
131939183
272041436
1177962436
499176036
206790179
1250979480
1488157403
76086111
465973696
607998896
1732986349
1642726357
177352548
2072729785
2049488214
564269794
1180570601
1810368154
1928114785
2017210484
1677424089
848205439
439776214
896075326
1594702889
1530365934
1196182092
371069233
1965202395

Ilustración 2: Ejecución del programa mostrando números pseudo-aleatorios enteros

Si se requiere generar números pseudo-aleatorios entre 0 y 1 este sería el código:

```
using System;

namespace Aleatorio {
    class Program {
        static void Main(string[] args) {
            Random azar = new Random();
            for (int cont=1; cont<=40; cont++) {
                Console.WriteLine(azar.NextDouble()); //Entre 0 y 1
            }
            Console.ReadKey();
        }
    }
}
```



```
C:\Users\engin\OneDrive\Proyecto\Libro\14. Genetico:
0,491903582816899
0,78392354761433
0,118758905268628
0,787189957586671
0,586141580988719
0,447253124065349
0,488729566097599
0,723023957909562
0,527134150512113
0,125202926399746
0,622522250107733
0,75637599302287
0,281274123713967
0,288572305947809
0,589599262731895
0,972111238153703
0,719896114300888
0,0576623659849457
0,827164331370576
0,0271517471536769
0,749864415149141
0,388483975729199
0,875038193014934
0,259717308571431
0,519328124597356
0,903254969000469
0,0174204581498264
0,0189656787640255
0,60365749690852
```

Ilustración 3: Ejecución del programa mostrando números pseudo-aleatorios entre 0 y 1

Cuidados a tener al usar los generadores

Obsérvese este código:

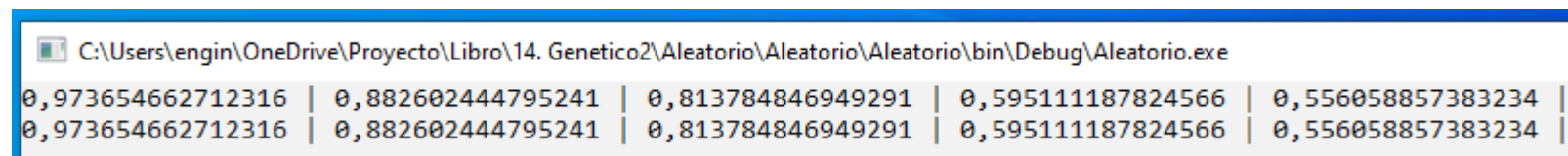
```
using System;

namespace Aleatorio {
    class Program {
        static void Main(string[] args) {
            Procedimiento1();
            Console.WriteLine(" ");
            Procedimiento2();
            Console.ReadKey();
        }

        static void Procedimiento1() {
            Random azar = new Random();
            for (int cont=1; cont<=5; cont++) {
                Console.Write(azar.NextDouble().ToString()+" | "); //Entre 0 y 1
            }
        }

        static void Procedimiento2() {
            Random aleatorio = new Random();
            for (int cont = 1; cont <= 5; cont++) {
                Console.Write(aleatorio.NextDouble().ToString() + " | "); //Entre 0 y 1
            }
        }
    }
}
```

Al ejecutar sucede esto:



```
C:\Users\engin\OneDrive\Proyecto\Libro\14. Genetico2\Aleatorio\Aleatorio\Aleatorio\bin\Debug\Aleatorio.exe
0,973654662712316 | 0,882602444795241 | 0,813784846949291 | 0,595111187824566 | 0,556058857383234 |
0,973654662712316 | 0,882602444795241 | 0,813784846949291 | 0,595111187824566 | 0,556058857383234 |
```

Ilustración 4: Se generan los mismos números pseudo-aleatorios

¿Cómo puede generar la misma serie de números pseudo-aleatorios si son instancias distintas y se ejecutan en procedimientos separados? La razón es que ambas instancias fueron inicializadas con la misma semilla (el reloj de la máquina), luego se obtienen los mismos números pseudo-aleatorios. La solución a esto es usar una sola instancia y enviarla como parámetro a cada procedimiento. Este sería el cambio en el código:

```
using System;

namespace Aleatorio {
    class Program {
        static void Main(string[] args) {
            Random azar = new Random();
            Procedimiento1(azar);
            Console.WriteLine(" ");
            Procedimiento2(azar);
            Console.ReadKey();
        }

        static void Procedimiento1(Random azar) {
            for (int cont=1; cont<=5; cont++) {
                Console.Write(azar.NextDouble().ToString()+" | "); //Entre 0 y 1
            }
        }

        static void Procedimiento2(Random aleatorio) {
            for (int cont = 1; cont <= 5; cont++) {
                Console.Write(aleatorio.NextDouble().ToString() + " | "); //Entre 0 y 1
            }
        }
    }
}
```

```
C:\Users\engin\OneDrive\Proyecto\Libro\14. Genetico2\Aleatorio\Aleatorio\Aleatorio\bin\Debug\Aleatorio.exe
0,271655532192278 | 0,582898035451257 | 0,837911303545307 | 0,358574838078848 | 0,920035819951462 |
0,283315527850443 | 0,0532248830670607 | 0,472374473452742 | 0,566421444325904 | 0,811177189839621 |
```

Ilustración 5: Números pseudo-aleatorios distintos

Mucho mejor. Luego es necesario hacer solo uso de una instancia de generador de números pseudo-aleatorios en todo el proyecto de algoritmos genéticos.

Iniciando con algoritmos genéticos

El problema

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. Igualmente se prueba.

El primer paso es generar cadenas al azar. Este sería el código:

```
using System;

namespace Cadenas {
    class Program {
        static void Main(string[] args) {
            //Único generador de números pseudo-aleatorios
            Random azar = new Random();

            //Imprime 10 cadenas al azar de 50 caracteres cada una
            for (int cont = 1; cont <= 10; cont++)
                Console.WriteLine(cont.ToString() + ": " + CadenaAzar(azar, 50));

            Console.ReadKey();
        }

        //Retorna una cadena al azar
        static string CadenaAzar(Random azar, int longitud) {
            string cad = "";
            for (int cont = 1; cont <= longitud; cont++) {
                cad += LetraAzar(azar);
            }
            return cad;
        }

        //Retorna una letra al azar
        static char LetraAzar(Random azar) {
            string Permitido = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ ";
            return Permitido[azar.Next(Permitido.Length)];
        }
    }
}
```

Un ejemplo de ejecución sería este:



```
C:\Users\engin\OneDrive\Proyecto\Libro\14. Genetico2\Cadenas\Cadenas\bin\Debug\Cadenas.exe
1: IeteKMKtMTBJyBIItlzapvD oXBIFxthuBFELhiYfWiQwNormSj
2: obptDffeJiefXuLhhTYXlbBWumOECjlxNoffeibzGdBPaRnxMG
3: fyowZmCu rtCEAZItKoFc RASHCNVD rTXdOiSpUFMiLgdAbRu
4: zDigEfHPFSLQAysbKIwmABwv rSguoIBWCdxIMtYgvoxjQmMue
5: vTGGUZhOhnpBvsl1UZuVTYpyefn mlefOpTPVrIfqvFgKG Xm
6: WosznxFhBPMiDYhiOgzWqEsoGyuiPAfrOnpSOpcxlNQTVgTIZB
7: TTzRKtXeTPcxuaJvwrJqwezecmDOzUzGNjpInTYDWWKwouPuQj
8: MDZJfuPwFgFymvjrnfxDz00vMdSKxjZpUNcPBipzxtwryzbdc
9: CpwOyoPmpvqYNIYHjsIgDNXfZffsxYnbOHWVDCwApDnPZBLlzl
10: ogvYBPsrQthyQrhplgncdJXiT BigywWUMKgIQfgEnMcsROWIL
```

Ilustración 6: Ejemplo de ejecución de generación de cadenas al azar

En la función LetraAzar se generan las letras del abecedario minúsculas, mayúsculas 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. Podría decirse que se usara el ASCII, pero de nuevo, si la cadena original tiene un carácter en otro idioma como el japonés (que no aparece en el ASCII) entonces jamás se encontraría una coincidencia.

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

Genetico00

```
//Generar cadenas al azar para acertar en cadenaOriginal
using System;

namespace Genetico00 {
    class Program {
        static void Main() {
            //Único generador de números pseudo-aleatorios
            Random azar = new Random();

            //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 num = 1; num <= 10000; num++) {
                string nuevaCadena = CadenaAzar(azar, longitud);
                if (nuevaCadena == cadenaOriginal)
                    Console.WriteLine("Acertó");
            }

            Console.WriteLine("Finaliza");
            Console.ReadKey();
        }

        //Retorna una cadena al azar
        static string CadenaAzar(Random azar, int longitud) {
            string cad = "";
            for (int cont = 1; cont <= longitud; cont++) {
                cad += LetraAzar(azar);
            }
            return cad;
        }

        //Retorna una letra al azar
        static char LetraAzar(Random azar) {
            string Permitido = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ ";
            return Permitido[azar.Next(Permitido.Length)];
        }
    }
}
```

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.

Los algoritmos genéticos solucionan ese problema e inspirados con la Teoría de la Evolución de las Especies, se agrega un nuevo ingrediente: la selección natural.

Algoritmo Genético: el operador Mutación

El algoritmo sería así:

```
Algoritmo Genético
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 íHmfGÓ
- 4: ÜkJñzU;mCoQEZe! ÄmúÖN¿Ghj zoá
- 5: tWsLótMRFhoB; íÄpkkDgYq; LrXëÓU
- 6: LFopAİéÓüYj!ëÖÄvÚMycCÄİİFPLÜ?
- 7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsíb
- 8: jOYnÚtAEH¿Ë;HVvPB ÄëÜáx¿fuÛch
- 9: ñróQEx;MqöüïEKÜiNTpRBóBWúofsö
- 10: ëWÄUPLúíiBVëÄÑLaómpKPÓ!İWXYÚQ

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

- 1: oTGBVáDgzD!óxëVoÓsbYAoVVRwkTj
- 2: Áh?JÄgÉmmÍ; !óQUbFtdfymNtNSjmF
- 3: rEVáa Z;oaÓC¿rxy?Äx¿pÑj íHmfGÓ
- 4: ÜkJñzU;mCoQEZe! ÄmúÖN¿Ghj zoá
- 5: tWsLótMRFhoB; íÄpkkDgYq; LrXëÓU
- 6: LFopAİéÓüYj!ëÖÄvÚMycCÄİİFPLÜ?
- 7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsíb
- 8: jOYnÚtAEH¿Ë;HVvPB ÄëÜáx¿fuÛch
- 9: ñróQEx;MqöüïEKÜiNTpRBóBWúofsö
- 10: ëWÄUPLúíiBVëÄÑLaómpKPÓ!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¿pÑj íHmfGÓ vs Esta es una prueba de texto Es 1

Evaluar: ñróQEx;MqöüïEKÜiNTpRBóBWúofsö vs Esta es una prueba de texto Es 0

El individuo ganador es rEVáa Z;oaÓC¿rxy?Äx¿pÑj íHmfGÓ con puntaje de 1, el otro pierde con un puntaje de 0

El individuo perdedor es eliminado de la población

1: oTGBVáDgzD!óxëVoÓsbYAoVVRwkTj
2: Áh?JÄgÉmmÍ;!óQUbFtdfymNtNSjmF
3: rEVáa Z;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: ñróQE×;MqöüiEKÜiNTpRDóBWúofse
10: ëWÄUPLúíiBVëÄÑLaómpKPÓ!İWXYÚQ

El mejor individuo se premia creando una copia de sí mismo

3: rEVáa Z;oaÓC¿rxy?Äx¿pÑj íHmfGÓ

Ahora esa copia se modifica en algún punto al azar

¿?: rEVáa Zh^hoaÓC¿rxy?Äx¿pÑj íHmfGÓ

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 íHmfGÓ
4: ÜkJñzU;mCoQEZe! ÄmúÖN¿Ghj zoá
5: tWsLótMRFhoB;íÄpkkDgYq;LrXëÓU
6: LFopAİéÓüYj!ëÖÄvÚMycCÄíİFPLÜ?
7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsíb
8: jOYnÚtAEH¿Ë;HVvPB ÄëÜáx¿fuÜch
9: rEVáa ZhoaÓC¿rxy?Äx¿pÑj íHmfGÓ
10: ëWÄUPLúíiBVëÄÑLaómpKPÓ!İWXYÚQ

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

Este es el código en el directorio **Genetico01**

Program.cs

```
using System;

namespace Genetico01 {
    class Program {
        static void Main() {
            string cadOriginal = "Estoy probando un algoritmo genético";

            Poblacion objPoblacion = new Poblacion();
            int TotalIndividuos = 50;
            int TotalCiclos = 30000;
            string MejorIndiv = objPoblacion.Proceso(cadOriginal, TotalIndividuos, TotalCiclos);

            //Muestra el individuo mejor adaptado
            Console.WriteLine(MejorIndiv);
            Console.ReadKey();
        }
    }
}
```

```

using System;
using System.Collections.Generic;

namespace Genetico01 {
    class Poblacion {
        private readonly List<string> Individuos; //Lista de individuos
        private readonly Random azar;

        public Poblacion() {
            Individuos = new List<string>();
            azar = new Random();
        }

        public string Proceso(string cadOriginal, int numIndividuos, int numCiclos) {
            Cadena objCad = new Cadena();

            //Crea la población con individuos generados al azar
            for (int cont = 1; cont <= numIndividuos; cont++) {
                Individuos.Add(objCad.CadenaAzar(azar, cadOriginal.Length));
            }

            //Proceso de algoritmo genético
            for (int ciclo = 1; ciclo <= numCiclos; ciclo++) {

                //Toma dos individuos al azar
                int indivA = azar.Next(Individuos.Count);
                int indivB;
                do {
                    indivB = azar.Next(Individuos.Count);
                } while (indivA == indivB); //Asegura que sean dos individuos distintos

                //Evalúa la adaptación de los dos individuos
                int valorIndivA = objCad.EvaluaCadena(cadOriginal, Individuos[indivA]);
                int valorIndivB = objCad.EvaluaCadena(cadOriginal, Individuos[indivB]);

                //Si individuo A está mejor adaptado que B entonces: Elimina B + Duplica A + Modifica duplicado
                if (valorIndivA > valorIndivB) {
                    Individuos[indivB] = objCad.MutaCadena(azar, Individuos[indivA]);
                }
                else { //Caso contrario: Elimina A + Duplica B + Modifica duplicado
                    Individuos[indivA] = objCad.MutaCadena(azar, Individuos[indivB]);
                }
            }

            //Después del ciclo, busca el mejor individuo adaptado de la población
            int individuoMejor = 0;
            int puntajeMejor = 0;
            for (int cont = 0; cont < Individuos.Count; cont++) {
                int valorIndiv = objCad.EvaluaCadena(cadOriginal, Individuos[cont]);
                if (valorIndiv > puntajeMejor) {
                    individuoMejor = cont;
                    puntajeMejor = valorIndiv;
                }
            }
            return Individuos[individuoMejor];
        }
    }
}

```

```

using System;

namespace Genetico01 {
    class Cadena {
        //Toma una cadena y le cambia alguna letra al azar
        public string MutaCadena(Random azar, string cadena) {
            int pos = azar.Next(cadena.Length);
            char[] cambia = cadena.ToCharArray();
            cambia[pos] = LetraAzar(azar);
            string nuevo = new string(cambia);
            return nuevo;
        }

        //Retorna en cuantos caracteres coincide cadA con cadB
        public int EvaluaCadena(string cadA, string cadB) {
            int acum = 0;
            for (int cont = 0; cont < cadA.Length; cont++) {
                if (cadA[cont] == cadB[cont]) acum++;
            }
            return acum;
        }

        //Genera una cadena al azar
        public string CadenaAzar(Random azar, int longitud) {
            string cad = "";
            for (int cont = 1; cont <= longitud; cont++) {
                cad += LetraAzar(azar).ToString();
            }
            return cad;
        }

        //Retorna un caracter al azar
        private char LetraAzar(Random azar) {
            string Permitido = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ áéúíóúÁÉÍÓÚÑñ¿?¡!äëïöüÄËÏÖÜ";
            return Permitido[azar.Next(Permitido.Length)];
        }
    }
}

```

Ejemplos de ejecuciones del programa:

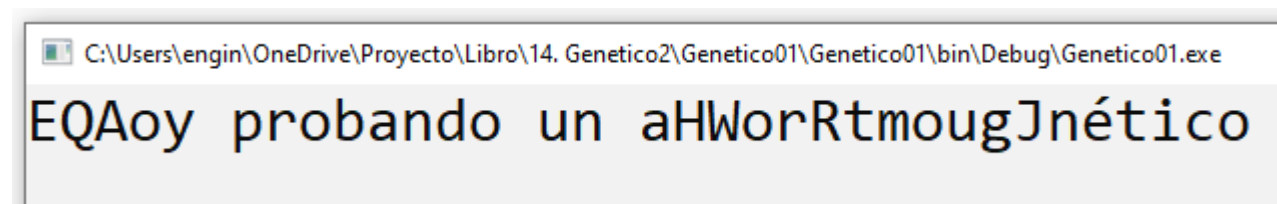


Ilustración 7: Ejecución del algoritmo genético. Con 10.000 ciclos

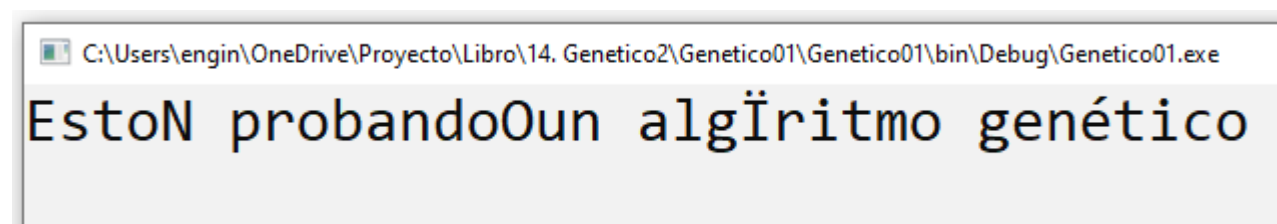


Ilustración 8: Ejecución del algoritmo genético. Con 20.000 ciclos

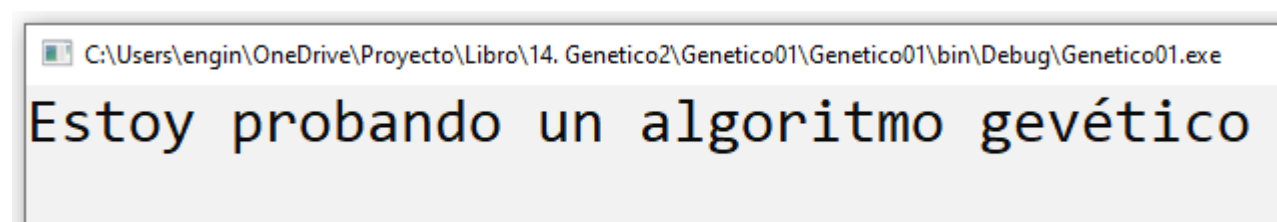


Ilustración 9: Ejecución del algoritmo genético. Con 30.000 ciclos

Algoritmo Genético: el operador Cruce

En la introducción dice que se inicia con un problema, en este caso es la cadena de destino:

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

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

- 1: oTGBVáDgzD!óxëVoÓsbYAoVVRwkTj
- 2: Áh?JÄgÉmmÍ;!óQUBftdfymNtNSjmF
- 3: rEVáaqZ;oaÓC¿rxy?Äx¿pÑj íHmfGÓ
- 4: ÜkJñzU;mCoQEZe! ÄmúÖN¿Ghj zoá
- 5: tWsLótMRFhoB;íÄpkkDgYq;LrXëÓU
- 6: LFopAİéÓüYj!ëÖÄvÚMycCÄİİFPLÜ?
- 7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsíb
- 8: jOYnÚtAEH¿Ë;HVvPB ÄëÜáx¿fuÜch
- 9: ñróQEx;MqöúïEKÜïNTpRBóBWúofsö
- 10: ëWÄUPLúíïBVëÄÑLaómpKPÓ!İWXYÚQ

Se seleccionan dos individuos al azar

- 1: oTGBVáDgzD!óxëVoÓsbYAoVVRwkTj
- 2: Áh?JÄgÉmmÍ;!óQUBftdfymNtNSjmF
- 3: rEVáaqZ;oaÓC¿rxy?Äx¿pÑj íHmfGÓ
- 4: ÜkJñzU;mCoQEZe! ÄmúÖN¿Ghj zoá
- 5: tWsLótMRFhoB;íÄpkkDgYq;LrXëÓU
- 6: LFopAİéÓüYj!ëÖÄvÚMycCÄİİFPLÜ?
- 7: ÑyérMQwlgMXaYTgúGsFfQÑIQBÜsíb
- 8: jOYnÚtAEH¿Ë;HVvPB ÄëÜáx¿fuÜch
- 9: ñróQEx;MqöúïEKÜïNTpRBóBWúofsö
- 10: ëWÄUPLúíïBVëÄÑLaómpKPÓ!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 padre y el resto el otro padre (en fondo verde son las partes de los padres que constituirán al hijo)

rEVáaqZ;oaÓC¿rxy?Äx¿pÑj íHmfGÓ y ñróQEx;MqöúïEKÜïNTpRBóBWúofsö

Hijo es al unir las dos piezas: rEVáax;MqöúïEKÜïNTpRBóBWúofsö

El hijo es evaluado y si es mejor que algún padre, entonces reemplaza a ese padre. El proceso se repite varias veces hasta que se encuentra el mejor individuo que soluciona el problema.

Este es el código ubicado en el directorio **Genetico02**

Program.cs

```
/* Algoritmo genético: Operador cruce */
using System;

namespace Genetico02 {
    class Program {
        static void Main() {
            string cadOriginal = "Estoy probando un algoritmo genético";

            Poblacion objPoblacion = new Poblacion();
            int TotalIndividuos = 2000;
            int TotalCiclos = 90000;
            string MejorIndiv = objPoblacion.Proceso(cadOriginal, TotalIndividuos, TotalCiclos);

            //Muestra el individuo mejor adaptado
            Console.WriteLine(MejorIndiv);
            Console.ReadKey();
        }
    }
}
```

```

/* Algoritmo genético: Operador cruce */
using System;
using System.Collections.Generic;

namespace Genetico02 {
    class Poblacion {
        private readonly List<string> Individuos; //Lista de individuos
        private readonly Random azar;

        public Poblacion() {
            Individuos = new List<string>();
            azar = new Random();
        }

        public string Proceso(string cadOriginal, int numIndividuos, int numCiclos) {
            Cadena objCad = new Cadena();

            //Crea la población con individuos generados al azar
            for (int cont = 1; cont <= numIndividuos; cont++) {
                Individuos.Add(objCad.CadenaAzar(azar, cadOriginal.Length));
            }

            //Proceso de algoritmo genético
            for (int ciclo = 1; ciclo <= numCiclos; ciclo++) {

                //Toma dos individuos al azar
                int indivA = azar.Next(Individuos.Count);
                int indivB;
                do {
                    indivB = azar.Next(Individuos.Count);
                } while (indivA == indivB); //Asegura que sean dos individuos distintos

                //Usa el operador cruce
                int posAzar = azar.Next(cadOriginal.Length);
                string parteA = Individuos[indivA].Substring(0, posAzar);
                string parteB = Individuos[indivB].Substring(posAzar);
                string HijoA = parteA + parteB;

                //Evalúa la adaptación de los individuos padres e hijos
                int valorIndivA = objCad.EvaluaCadena(cadOriginal, Individuos[indivA]);
                int valorIndivB = objCad.EvaluaCadena(cadOriginal, Individuos[indivB]);
                int valorHijoA = objCad.EvaluaCadena(cadOriginal, HijoA);

                //Si los hijos son mejores que los padres, entonces los reemplaza
                if (valorHijoA > valorIndivA) Individuos[indivA] = HijoA;
                else if (valorHijoA > valorIndivB) Individuos[indivB] = HijoA;
            }

            //Después del ciclo, busca el mejor individuo adaptado de la población
            int individuoMejor = 0;
            int puntajeMejor = 0;
            for (int cont = 0; cont < Individuos.Count; cont++) {
                int valorIndiv = objCad.EvaluaCadena(cadOriginal, Individuos[cont]);
                if (valorIndiv > puntajeMejor) {
                    individuoMejor = cont;
                    puntajeMejor = valorIndiv;
                }
            }
            return Individuos[individuoMejor];
        }
    }
}

```



```

/* Algoritmo genético: Operador cruce */
using System;

namespace Genetico02 {
    class Cadena {
        //Toma una cadena y le cambia alguna letra al azar
        public string MutaCadena(Random azar, string cadena) {
            int pos = azar.Next(cadena.Length);
            char[] cambia = cadena.ToCharArray();
            cambia[pos] = LetraAzar(azar);
            string nuevo = new string(cambia);
            return nuevo;
        }

        //Retorna en cuantos caracteres coincide cadA con cadB
        public int EvaluaCadena(string cadA, string cadB) {
            int acum = 0;
            for (int cont = 0; cont < cadA.Length; cont++) {
                if (cadA[cont] == cadB[cont]) acum++;
            }
            return acum;
        }

        //Genera una cadena al azar
        public string CadenaAzar(Random azar, int longitud) {
            string cad = "";
            for (int cont = 1; cont <= longitud; cont++) {
                cad += LetraAzar(azar).ToString();
            }
            return cad;
        }

        //Retorna un caracter al azar
        private char LetraAzar(Random azar) {
            string Permitido = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ áéúíóúÁÉÍÓÚÑñ¿?¡!äëïöüĂĖİÖÜ";
            return Permitido[azar.Next(Permitido.Length)];
        }
    }
}

```

Resultado arrojado por este programa:

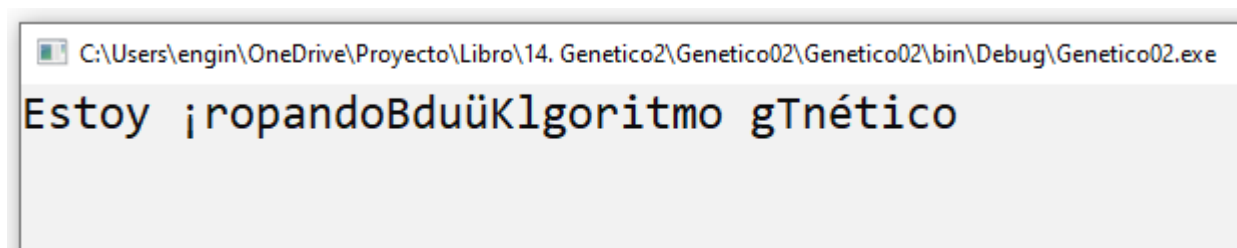


Ilustración 10: Ejecución del algoritmo genético con el operador cruce. Uso de 50.000 ciclos.

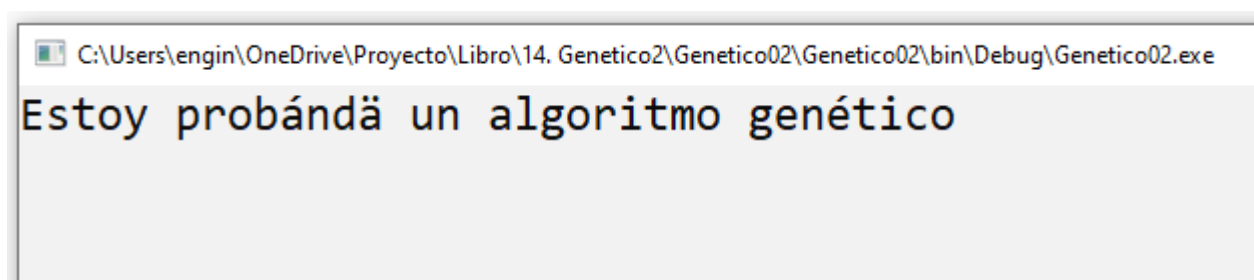


Ilustración 11: Ejecución del algoritmo genético con el operador cruce. Uso de 70.000 ciclos.

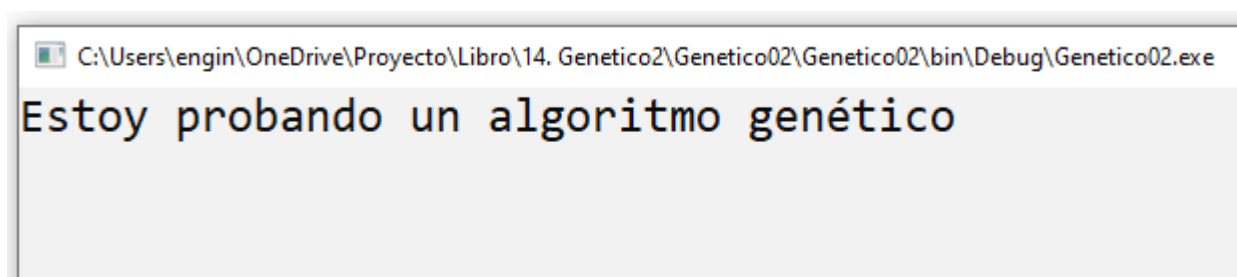


Ilustración 12: Ejecución del algoritmo genético con el operador cruce. Uso de 90.000 ciclos.

Algoritmo Genético: 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. Este sería el código ubicado en el directorio **Genetico03**

Program.cs

```
/* Algoritmo genético: Operador cruce y mutación */
using System;

namespace Genetico03 {
    class Program {
        static void Main() {
            string cadOriginal = "Estoy probando un algoritmo genético";

            Poblacion objPoblacion = new Poblacion();
            int TotalIndividuos = 2000;
            int TotalCiclos = 100000;
            string MejorIndiv = objPoblacion.Proceso(cadOriginal, TotalIndividuos, TotalCiclos);

            //Muestra el individuo mejor adaptado
            Console.WriteLine(MejorIndiv);
            Console.ReadKey();
        }
    }
}
```

```

/* Algoritmo genético: Operador cruce y mutación */
using System;
using System.Collections.Generic;

namespace Genetico03 {
    class Poblacion {
        private readonly List<string> Individuos; //Lista de individuos
        private readonly Random azar;

        public Poblacion() {
            Individuos = new List<string>();
            azar = new Random();
        }

        public string Proceso(string cadOriginal, int numIndividuos, int numCiclos) {
            Cadena objCad = new Cadena();

            //Crea la población con individuos generados al azar
            for (int cont = 1; cont <= numIndividuos; cont++) {
                Individuos.Add(objCad.CadenaAzar(azar, cadOriginal.Length));
            }

            //Proceso de algoritmo genético
            for (int ciclo = 1; ciclo <= numCiclos; ciclo++) {

                //Toma dos individuos al azar
                int indivA = azar.Next(Individuos.Count);
                int indivB;
                do {
                    indivB = azar.Next(Individuos.Count);
                } while (indivA == indivB); //Asegura que sean dos individuos distintos

                //Usa el operador cruce
                int posAzar = azar.Next(cadOriginal.Length);
                string parteA = Individuos[indivA].Substring(0, posAzar);
                string parteB = Individuos[indivB].Substring(posAzar);
                string HijoA = parteA + parteB;

                //Además muta el hijo
                HijoA = objCad.MutaCadena(azar, HijoA);

                //Evalúa la adaptación de los individuos padres e hijos
                int valorIndivA = objCad.EvaluaCadena(cadOriginal, Individuos[indivA]);
                int valorIndivB = objCad.EvaluaCadena(cadOriginal, Individuos[indivB]);
                int valorHijoA = objCad.EvaluaCadena(cadOriginal, HijoA);

                //Si los hijos son mejores que los padres, entonces los reemplaza
                if (valorHijoA > valorIndivA) Individuos[indivA] = HijoA;
                else if (valorHijoA > valorIndivB) Individuos[indivB] = HijoA;
            }

            //Después del ciclo, busca el mejor individuo adaptado de la población
            int individuoMejor = 0;
            int puntajeMejor = 0;
            for (int cont = 0; cont < Individuos.Count; cont++) {
                int valorIndiv = objCad.EvaluaCadena(cadOriginal, Individuos[cont]);
                if (valorIndiv > puntajeMejor) {
                    individuoMejor = cont;
                    puntajeMejor = valorIndiv;
                }
            }
            return Individuos[individuoMejor];
        }
    }
}

```

```
/* Algoritmo genético: Operador cruce y mutación */
using System;

namespace Genetico03 {
    class Cadena {
        //Toma una cadena y le cambia alguna letra al azar
        public string MutaCadena(Random azar, string cadena) {
            int pos = azar.Next(cadena.Length);
            char[] cambia = cadena.ToCharArray();
            cambia[pos] = LetraAzar(azar);
            string nuevo = new string(cambia);
            return nuevo;
        }

        //Retorna en cuantos caracteres coincide cadA con cadB
        public int EvaluaCadena(string cadA, string cadB) {
            int acum = 0;
            for (int cont = 0; cont < cadA.Length; cont++) {
                if (cadA[cont] == cadB[cont]) acum++;
            }
            return acum;
        }

        //Genera una cadena al azar
        public string CadenaAzar(Random azar, int longitud) {
            string cad = "";
            for (int cont = 1; cont <= longitud; cont++) {
                cad += LetraAzar(azar).ToString();
            }
            return cad;
        }

        //Retorna un caracter al azar
        private char LetraAzar(Random azar) {
            string Permitido = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ áéúíóúÁÉÍÓÚÑñ¿?¡!äëïöüÄËÏÖÜ";
            return Permitido[azar.Next(Permitido.Length)];
        }
    }
}
```

Al ejecutar se obtiene esta salida:

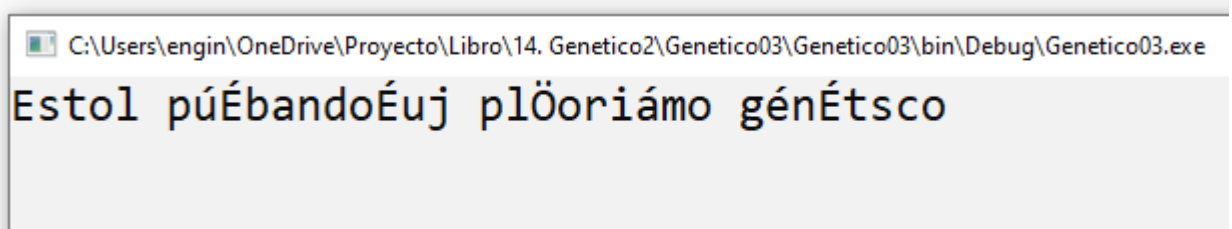


Ilustración 13: Ejecución del algoritmo genético: operador cruce y mutación. Uso de 50.000 ciclos

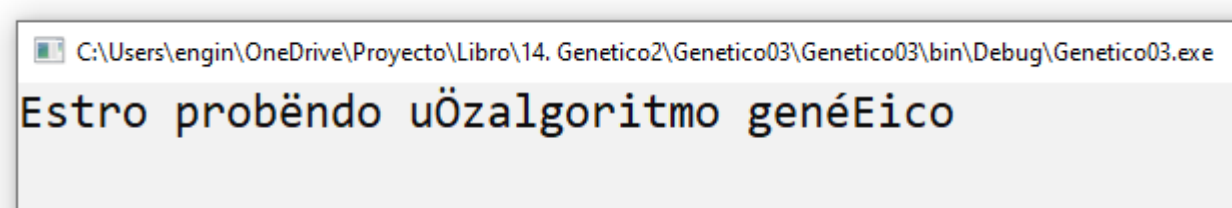


Ilustración 14: Ejecución del algoritmo genético: operador cruce y mutación. Uso de 70.000 ciclos

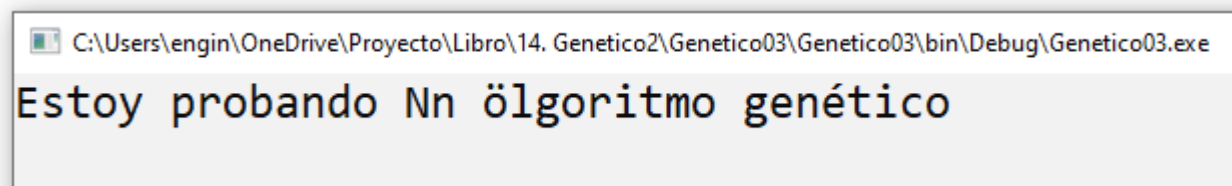


Ilustración 15: Ejecución del algoritmo genético: operador cruce y mutación. Uso de 100.000 ciclos

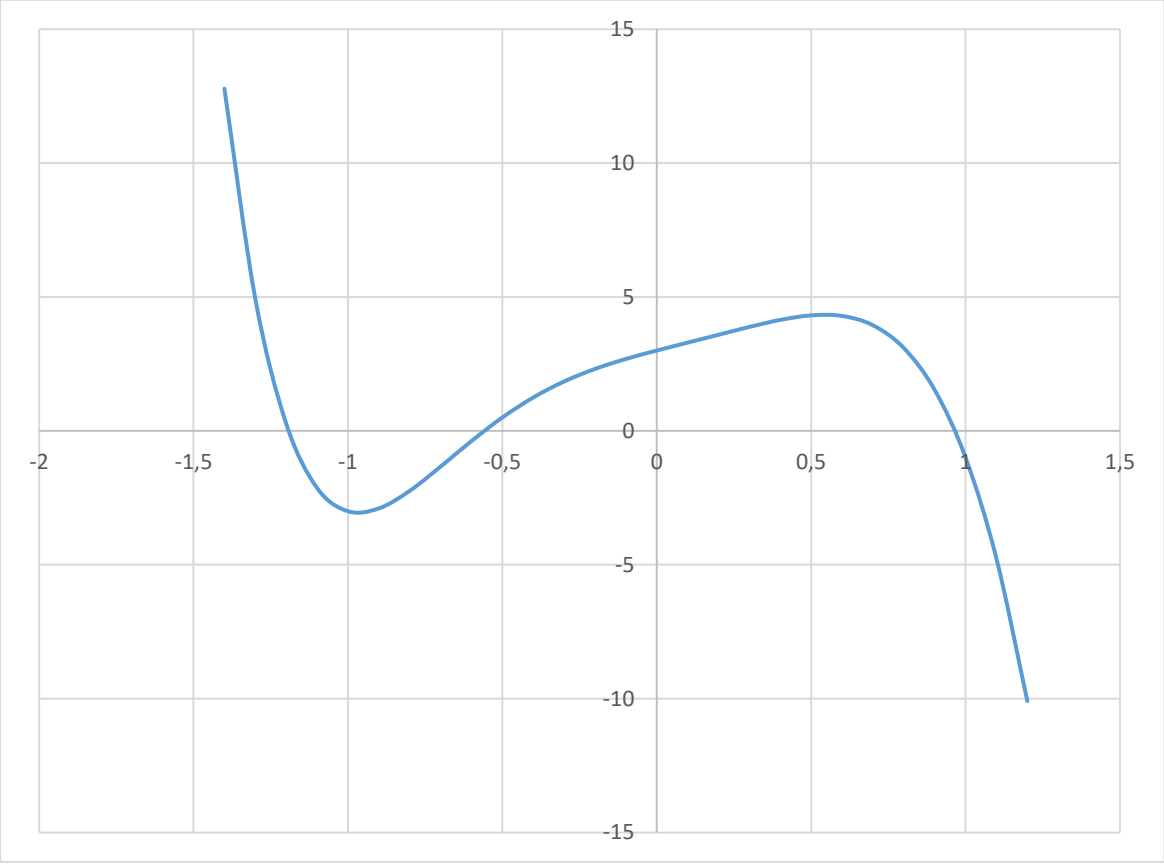
Consideraciones

1. Hay muchas partes en las cuáles esos dos algoritmos se pueden cambiar y observar si son mejores o más rápidos.
2. En el algoritmo de mutación pueden hacerse estos cambios:
 - a. Aumentar o disminuir el número de ciclos.
 - b. Validar que cuando logre dar con la cadena, el programa se detenga.
 - c. Seleccionar tres o más individuos y que compitan entre estos para ver cuál es el mejor para generar un hijo.
 - d. Se puede escoger sólo un individuo, duplicarlo y mutarlo, repetir hasta que la copia mutada sea mejor que el progenitor y reemplazar el progenitor.
 - e. Mutar dos o más partes del hijo.
 - f. Se puede optimizar, almacenando el valor de adaptación y así no es necesario calcularlo constantemente.
 - g. Siguiendo el punto anterior, 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.
3. En el algoritmo de cruce pueden hacerse estos cambios:
 - a. Aumentar o disminuir el número de ciclos.
 - b. Validar que cuando logre dar con la cadena, el programa se detenga.
 - c. Seleccionar tres y que se logre un cruce entre estos con dos puntos de corte.
 - d. Generar dos o más hijos.
4. En el algoritmo de cruce y mutación se pueden aplicar los cambios propuestos de los dos anteriores
5. Los tres algoritmos de mutación o cruce o combinados tienen como punto configurable el tamaño de la población, entre más grande sea la población, hay mayor variedad en los individuos, pero se tarda más el proceso de encontrar el mejor individuo. Con una población pequeña se corre el riesgo que no haya variedad suficiente para dar con el mejor individuo.
6. ¿Por qué mantener la población en un número constante y no permitir que crezca o disminuya?
7. Hay partes que se pueden optimizar del código haciendo uso de APIs más rápidas en C#.
8. También podría cambiarse el generador de números pseudo-aleatorios.
9. Un paso sería hacer pruebas sobre cuál algoritmo logra el más rápido acercamiento.

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**, nos piden encontrar en cuál valor de "**x**", se obtiene el máximo valor de "**y**" en ese rango **[a,b]**. Hallarlo sería 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 podemos aplicar los algoritmos genéticos.

Por ejemplo, tenemos esta curva:



Queremos hallar el valor de X que obtenga el mayor valor de Y en el rango [0, 1). La curva es:

$$Y = 2 * X^6 - 7 * X^5 - 6 * X^4 + 5 * X^3 - X^2 + 3 * X + 3$$

El primer paso con los algoritmos genéticos 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 entre [0,1) y para este ejemplo, tenemos suerte, porque es el valor que nos arrojaría un generador de números pseudo-aleatorios como el Random.

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#, donde podemos ver que se usa la misma mecánica:

1. Generar una población de individuos al azar (cada individuo es un número de tipo double).
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 restandole 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 no está optimizado y tiene un número fijo de ciclos porque el valor de X que genere el máximo Y es un número irracional, así que sólo se puede aproximar. El código está ubicado en el directorio **Genetico04**:

Program.cs

```
/* Búsqueda del máximo. Algoritmo genético: Operador mutación */
using System;

namespace Genetico04 {
    class Program {
        static void Main() {
            Poblacion objPoblacion = new Poblacion();
            int TotalIndividuos = 50;
            int TotalCiclos = 1000;
            double MejorIndiv = objPoblacion.Proceso(TotalIndividuos, TotalCiclos);

            //Muestra el individuo mejor adaptado
            Console.WriteLine("X = " + MejorIndiv.ToString());
            Console.ReadKey();
        }
    }
}
```

```

/* Búsqueda del máximo de una función. Algoritmo genético: Operador mutación */
using System;
using System.Collections.Generic;

namespace Genetico04 {
    class Poblacion {
        private readonly List<double> Individuos; //Lista de individuos
        private readonly Random azar;

        public Poblacion() {
            Individuos = new List<double>();
            azar = new Random();
        }

        public double Proceso(int numIndividuos, int numCiclos) {

            //Crea la población con individuos generados al azar
            for (int cont = 1; cont <= numIndividuos; cont++) {
                Individuos.Add(azar.NextDouble());
            }

            //Proceso de algoritmo genético
            for (int ciclo = 1; ciclo <= numCiclos; ciclo++) {

                //Toma dos individuos al azar
                int indivA = azar.Next(Individuos.Count);
                int indivB;
                do {
                    indivB = azar.Next(Individuos.Count);
                } while (indivA == indivB); //Asegura que sean dos individuos distintos

                //Evalúa la adaptación de los dos individuos
                double valorIndivA = EvaluaEcuacion(Individuos[indivA]);
                double valorIndivB = EvaluaEcuacion(Individuos[indivB]);

                //Si individuo A está mejor adaptado que B entonces: Elimina B + Duplica A + Modifica duplicado
                if (valorIndivA > valorIndivB) {
                    Individuos[indivB] = MutaValor(Individuos[indivA]);
                }
                else { //Caso contrario: Elimina A + Duplica B + Modifica duplicado
                    Individuos[indivA] = MutaValor(Individuos[indivB]);
                }
            }

            //Después del ciclo, busca el mejor individuo adaptado de la población
            int individuoMejor = 0;
            double puntajeMejor = double.MinValue;
            for (int cont = 0; cont < Individuos.Count; cont++) {
                double valorIndiv = EvaluaEcuacion(Individuos[cont]);
                if (valorIndiv > puntajeMejor) {
                    individuoMejor = cont;
                    puntajeMejor = valorIndiv;
                }
            }
            return Individuos[individuoMejor];
        }

        //Retorna el valor de Y de la ecuación dado el valor de X
        private double EvaluaEcuacion(double x) {
            double y = 2 * Math.Pow(x, 6) - 7 * Math.Pow(x, 5);
            y = y - 6 * Math.Pow(x, 4) + 5 * Math.Pow(x, 3);
            y = y - x * x + 3 * x + 3;
            return y;
        }

        //Muta un valor sumando o restando al azar un valor del tipo 0.1, 0.01, 0.001
        private double MutaValor(double x) {
            int potencia = azar.Next(7) + 1;
            double cambio = 1 / Math.Pow(10, potencia);
            if (azar.NextDouble() < 0.5) {
                return x + cambio;
            }
            return x - cambio;
        }
    }
}

```

Un ejemplo de su ejecución

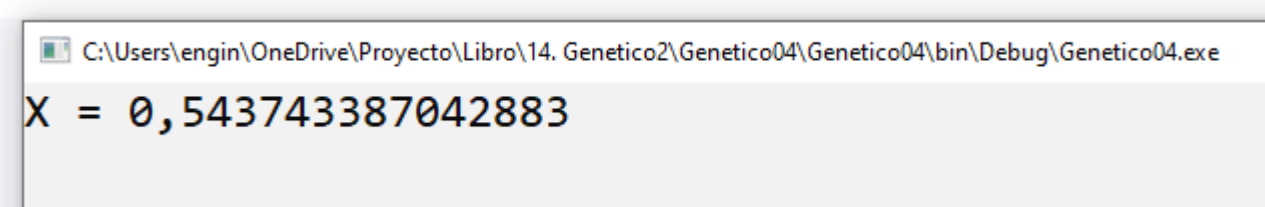


Ilustración 16: Buscando el valor de X donde es el máximo Y de una función

Comparando resultados con WolframAlpha [2]

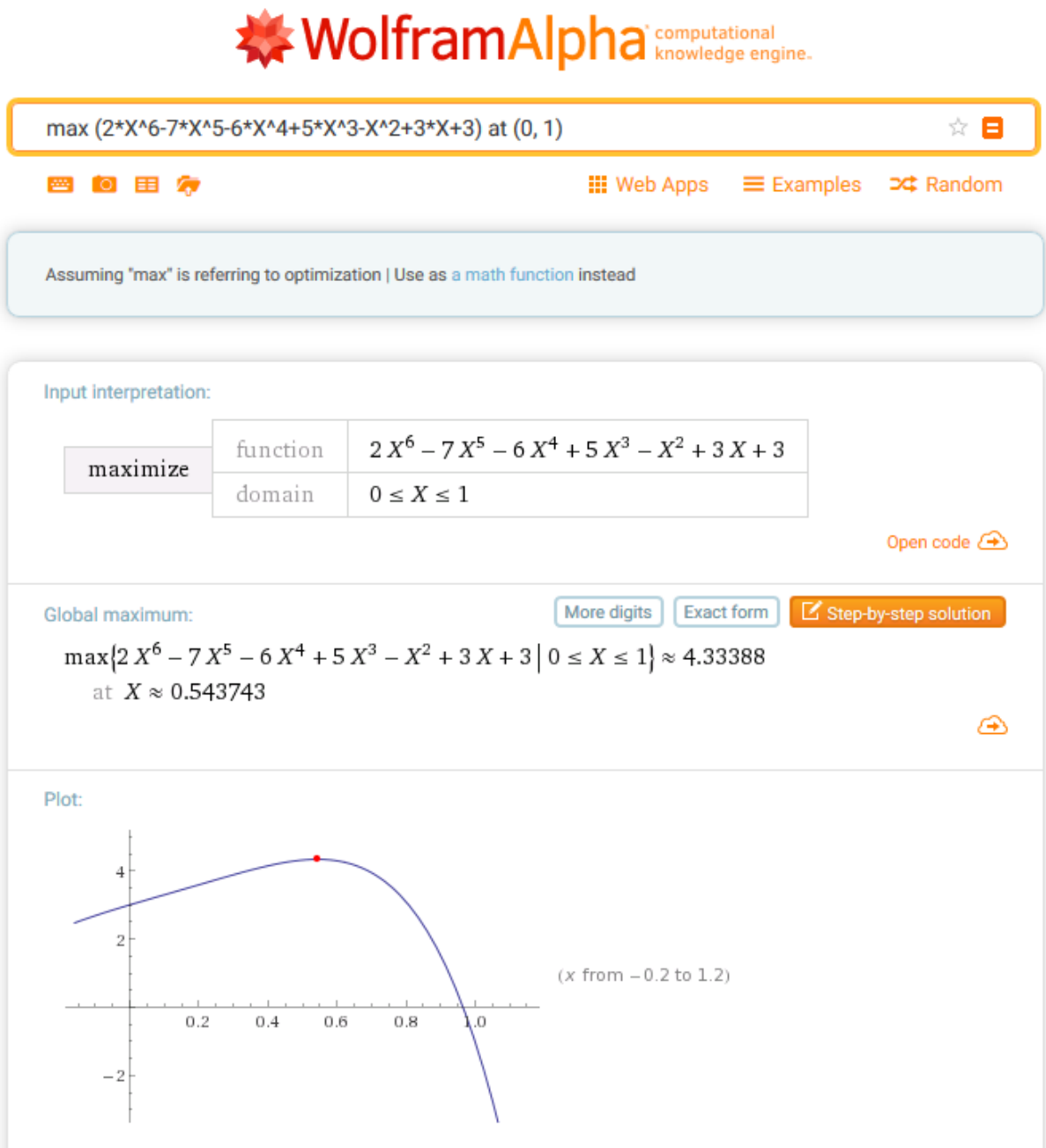


Ilustración 17: Búsqueda del valor X para un máximo Y en un determinado rango

El algoritmo genético logra encontrar el máximo con muy buena precisión.

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. Aunque el algoritmo funciona, tiene puntos débiles cuando se requiera buscar el valor de X en un rango más amplio que el de [0,1). Debe haber una forma para ampliar ese rango.

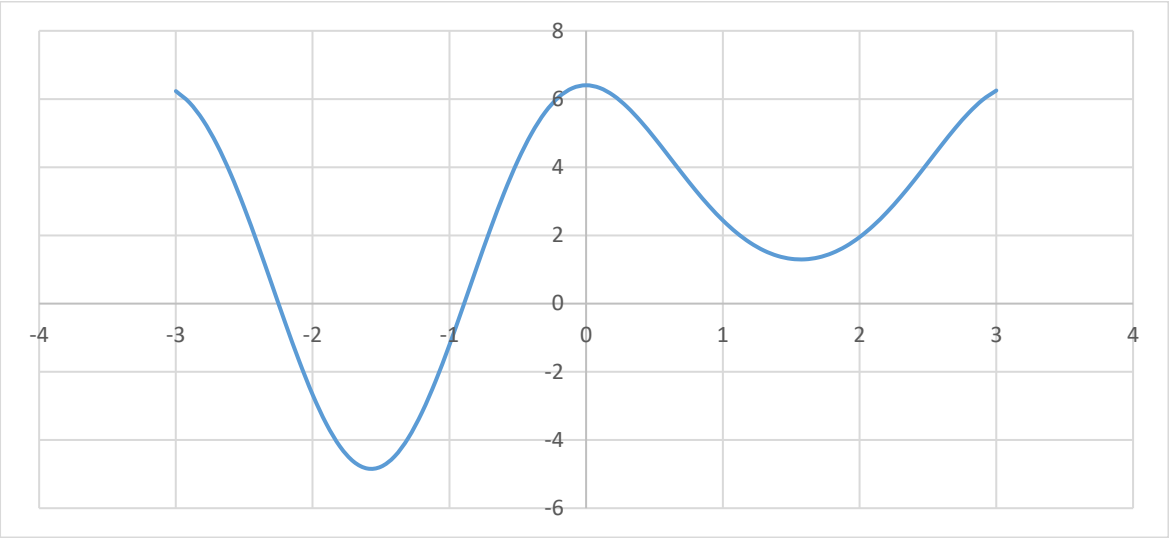
En los seres vivos, las instrucciones para su desarrollo y funcionamiento están en el ADN. Esa información genética es el genotipo. La interpretación de ese genotipo es el fenotipo [3]. Un ejemplo grosso modo:

Genotipo	Fenotipo
ACCTGAACTTGGCCAATGGCCTAACCTG	Forma del pico de un ave

En algoritmos genéticos 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, tenemos la siguiente función:

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



Nos piden encontrar el valor de "x" entre -3 y 2, donde se obtenga el mínimo valor de "y". A ojo, viendo la gráfica sabemos que es aproximadamente x=-1.6, pero eso no es preciso. Se requiere precisión y este sería el proceso:

Paso 1

Definimos 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 nuestro ejemplo, el valor mínimo -3 será representado por "000000000" y el valor máximo 2 será representado por "111111111". ¿Cuántas combinaciones hay? Es 2^{total_cifras}, luego en este caso concreto sería 2⁹ = 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 debemos hacer la conversión valor binario a valor decimal aquí!

Paso 2

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

x = RangoMinimo + valorbinarioadecimal * (RangoMáximo - RangoMínimo) / (2^{TotalCifras} - 1)

Probemos con "000000000" (0 en decimal)

x = -3 + 0 * (2 - -3) / (2⁹ - 1) = -3 + 0 * 5 / 511 = -3 + 0 = -3

Probemos con "111111111" (511 en decimal)

x = -3 + 511 * (2 - -3) / (2⁹ - 1) = -3 + 511 * 5 / 511 = -3 + 5 = 2

Probemos con "011011101" (221 en decimal)

x = -3 + 221 * (2 - -3) / (2⁹ - 1) = -3 + 221 * 5 / 511 = -0,837573385518591

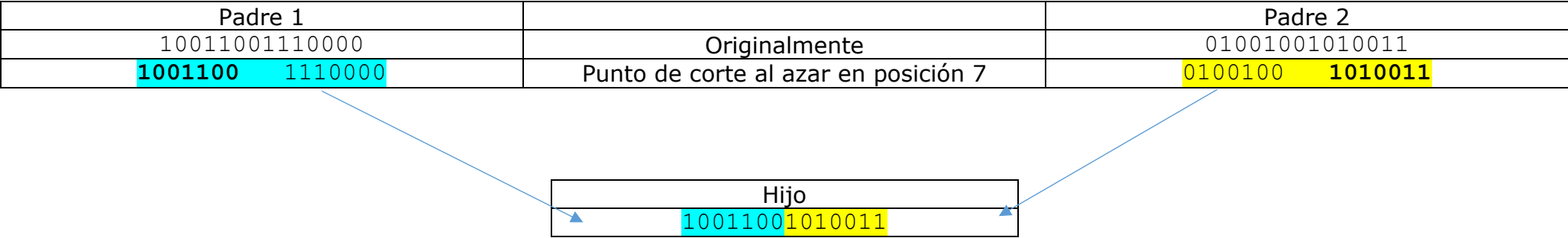
Paso 3

La representación binaria de un individuo nos da una ventaja y es 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. Veamos un ejemplo:

Individuo	10011001110000
Posición al azar 9 que es	10011001 1 10000
Cambie 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. Veamos un ejemplo



Paso 5

¿Cómo el lenguaje de programación elegido para implementar el algoritmo genético puede operar con valores binarios? ¿tendrá operadores de fácil uso? ¿El desempeño será bueno?

En este libro usaremos un enfoque que podríamos decir que es universal y es haciendo uso de una cadena (string), donde en cada posición de la cadena se pone o un 1 o un 0, veamos:

1	0	0	1	1	0	0	1	1	1	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---

Paso 6

Definir una población. Los algoritmos genéticos inician con una población inicial de individuos generados al azar. Siguiendo el enfoque anterior, la solución a esto es una lista de cadenas.

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 menor 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 menor Y de la población
  Imprimir individuo
Fin
```

Este es el código en C# ubicado en el directorio **Genetico05**:

Program.cs

```
/* Algoritmo genético: Operador mutación */
using System;

namespace Genetico05 {
  class Program {
    static void Main() {
      double minValorX = -3;
      double maxValorX = 2;
      Ecuacion objEcuacion = new Ecuacion();
      objEcuacion.Rango(minValorX, maxValorX);

      Poblacion objPoblacion = new Poblacion();
      int TamanoIndividuo = 30;
      int TotalIndividuos = 50;
      int TotalCiclos = 100000;
      double MejorIndiv = objPoblacion.Proceso(objEcuacion, TamanoIndividuo, TotalIndividuos, TotalCiclos);

      //Muestra el individuo mejor adaptado
      Console.WriteLine(MejorIndiv);
      Console.ReadKey();
    }
  }
}
```

```

/* Buscar valor de X. Algoritmo genético: Operador mutación */
using System;
using System.Collections.Generic;

namespace Genetico05 {
    class Poblacion {
        private readonly List<string> Individuos; //Lista de individuos
        private readonly Random azar;

        public Poblacion() {
            Individuos = new List<string>();
            azar = new Random();
        }

        public double Proceso(Ecuacion objEcuacion, int TamanoIndividuo, int numIndividuos, int numCiclos) {

            //Crea la población con individuos generados al azar
            for (int cont = 1; cont <= numIndividuos; cont++) {
                Individuos.Add(BooleanoAzar(TamanoIndividuo));
            }

            //Proceso de algoritmo genético
            for (int ciclo = 1; ciclo <= numCiclos; ciclo++) {

                //Toma dos individuos al azar
                int indivA = azar.Next(Individuos.Count);
                int indivB;
                do {
                    indivB = azar.Next(Individuos.Count);
                } while (indivA == indivB); //Asegura que sean dos individuos distintos

                //Evalúa la adaptación de los dos individuos
                double valorIndivA = objEcuacion.ValorY(Individuos[indivA]);
                double valorIndivB = objEcuacion.ValorY(Individuos[indivB]);

                //Si individuo A está mejor adaptado que B entonces: Elimina B + Duplica A + Modifica duplicado
                if (valorIndivA < valorIndivB) {
                    CopiayMuta(indivB, indivA);
                }
                else { //Caso contrario: Elimina A + Duplica B + Modifica duplicado
                    CopiayMuta(indivA, indivB);
                }
            }

            //Después del ciclo, busca el mejor individuo adaptado de la población
            int individuoMejor = 0;
            double MenorValorY = double.MaxValue;
            for (int cont = 0; cont < Individuos.Count; cont++) {
                double valorIndiv = objEcuacion.ValorY(Individuos[cont]);
                if (valorIndiv < MenorValorY) {
                    individuoMejor = cont;
                    MenorValorY = valorIndiv;
                }
            }
            return objEcuacion.ValorX(Individuos[individuoMejor]);
        }

        //Genera un individuo booleano al azar
        private string BooleanoAzar(int Tamano) {
            string numero = "";
            for (int cont = 1; cont <= Tamano; cont++) {
                if (azar.NextDouble() < 0.5)
                    numero += "1";
                else
                    numero += "0";
            }
            return numero;
        }

        //Copia el ganador sobre el perdedor y modifica la copia
        private void CopiayMuta(int indPerdedor, int indGanador) {
            //Copia el individuo ganador sobre el perdedor
            Individuos[indPerdedor] = Individuos[indGanador];

            //Muta la copia
            char[] numeros = Individuos[indPerdedor].ToCharArray();

```

```

        int pos = azar.Next(Individuos[indPerdedor].Length);
        if (numeros[pos] == '0')
            numeros[pos] = '1';
        else
            numeros[pos] = '0';
        Individuos[indPerdedor] = new string(numeros);
    }
}
}

```

Ecuacion.cs

```

/* Buscar valor de X. Algoritmo genético: Operador mutación */
using System;

namespace Genetico05 {
    class Ecuacion {
        private double minX, maxX;

        //El rango de búsqueda en X para encontrar el mínimo Y
        public void Rango(double minX, double maxX) {
            this.minX = minX;
            this.maxX = maxX;
        }

        //El individuo en binario se convierte a real y
        //retorna el valor que genera la ecuación
        public double ValorY(string Individuo) {
            double x = ValorX(Individuo);
            double y = -1.78 * Math.Pow(Math.Sin(x), 2);
            y += 6.40 * Math.Pow(Math.Cos(x), 2);
            y += 3.07 * Math.Pow(Math.Sin(x), 3);
            return y;
        }

        //Convierte la representación binaria en dato tipo real
        public double ValorX(string Individuo) {
            double multiplica = (maxX - minX);
            multiplica /= (Math.Pow(2, Individuo.Length) - 1);
            int numero = Convert.ToInt32(Individuo, 2);
            return minX + numero * multiplica;
        }
    }
}

```

Ejecución del programa:

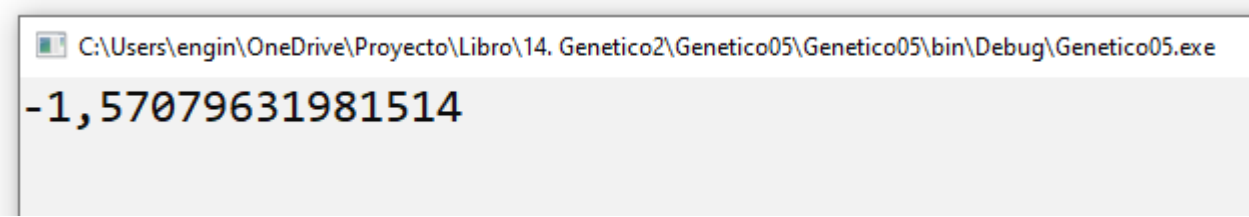


Ilustración 18: Ejecución del programa

Una aproximación muy buena. Comparándola con lo obtenido con Wolfram Alpha:

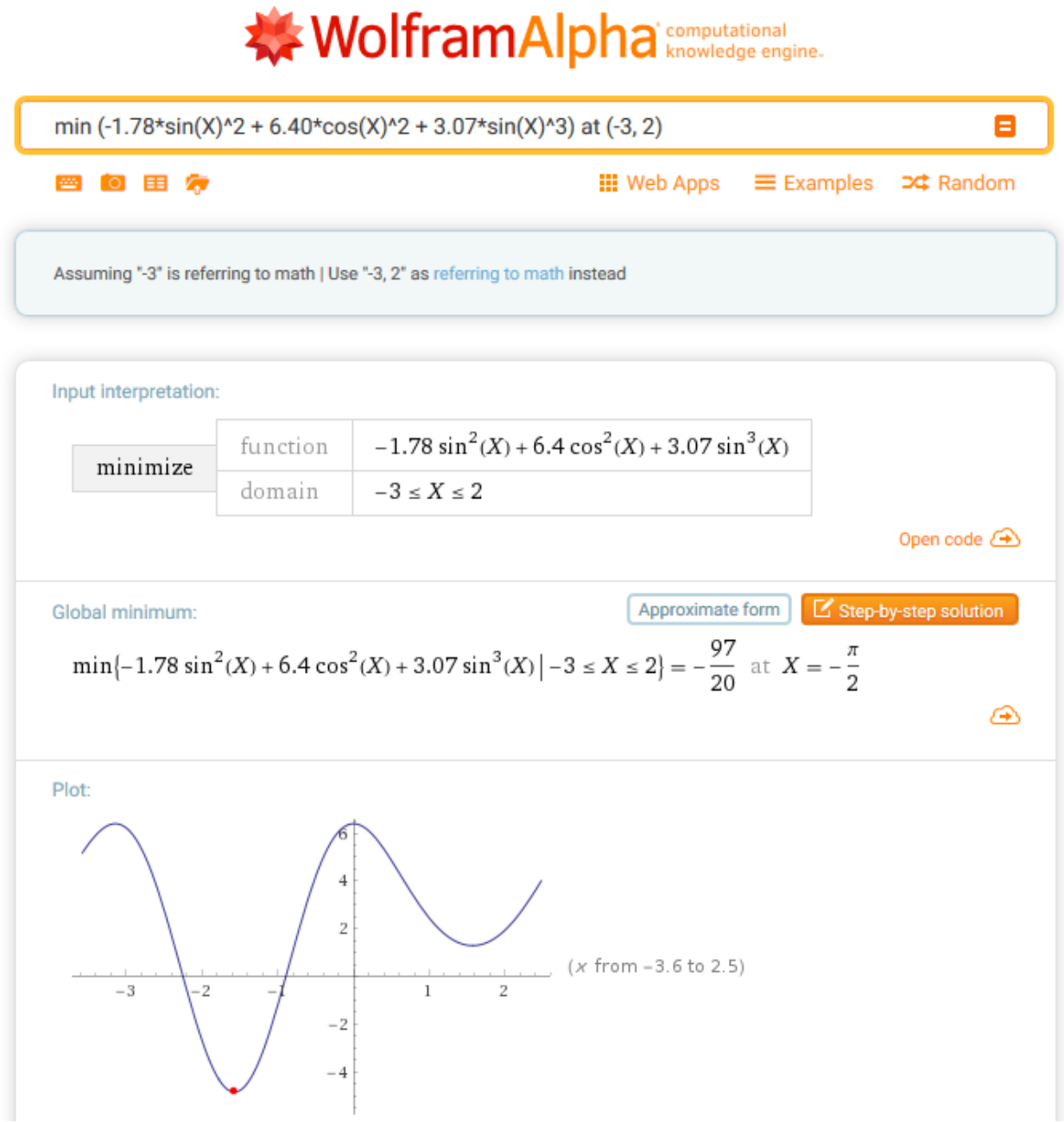


Ilustración 19: Buscando el mínimo de una función con Wolfram Alpha

$$X = -\frac{\pi}{2} = -1,5707963267948966192313216916398$$

Operador cruce

Se hace un cambio a la clase Poblacion, este sería el código en el directorio **Genetico06**:

Program.cs

```
/* Algoritmo genético: Operador cruce */
using System;

namespace Genetico06 {
    class Program {
        static void Main() {
            double minValorX = -3;
            double maxValorX = 2;
            Ecuacion objEcuacion = new Ecuacion();
            objEcuacion.Rango(minValorX, maxValorX);

            Poblacion objPoblacion = new Poblacion();
            int TamanoIndividuo = 30;
            int TotalIndividuos = 50;
            int TotalCiclos = 100000;
            double MejorIndiv = objPoblacion.Proceso(objEcuacion, TamanoIndividuo, TotalIndividuos, TotalCiclos);

            //Muestra el individuo mejor adaptado
            Console.WriteLine(MejorIndiv);
            Console.ReadKey();
        }
    }
}
```

```

/* Buscar valor de X. Algoritmo genético: Operador cruce */
using System;
using System.Collections.Generic;

namespace Genetico06 {
    class Poblacion {
        private readonly List<string> Individuos; //Lista de individuos
        private readonly Random azar;

        public Poblacion() {
            Individuos = new List<string>();
            azar = new Random();
        }

        public double Proceso(Ecuacion objEcuacion, int TamanoIndividuo, int numIndividuos, int numCiclos) {

            //Crea la población con individuos generados al azar
            for (int cont = 1; cont <= numIndividuos; cont++) {
                Individuos.Add(BooleanoAzar(TamanoIndividuo));
            }

            //Proceso de algoritmo genético
            for (int ciclo = 1; ciclo <= numCiclos; ciclo++) {

                //Toma dos individuos al azar
                int indivA = azar.Next(Individuos.Count);
                int indivB;
                do {
                    indivB = azar.Next(Individuos.Count);
                } while (indivA == indivB); //Asegura que sean dos individuos distintos

                //Usa el operador cruce
                int posAzar = azar.Next(Individuos[indivA].Length);
                string parteA = Individuos[indivA].Substring(0, posAzar);
                string parteB = Individuos[indivB].Substring(posAzar);
                string HijoA = parteA + parteB;

                //Evalúa la adaptación de los dos individuos
                double valorIndivA = objEcuacion.ValorY(Individuos[indivA]);
                double valorIndivB = objEcuacion.ValorY(Individuos[indivB]);
                double valorHijoA = objEcuacion.ValorY(HijoA);

                //Si los hijos son mejores que los padres, entonces los reemplaza
                if (valorHijoA < valorIndivA) Individuos[indivA] = HijoA;
                else if (valorHijoA < valorIndivB) Individuos[indivB] = HijoA;

            }

            //Después del ciclo, busca el mejor individuo adaptado de la población
            int individuoMejor = 0;
            double MenorValorY = double.MaxValue;
            for (int cont = 0; cont < Individuos.Count; cont++) {
                double valorIndiv = objEcuacion.ValorY(Individuos[cont]);
                if (valorIndiv < MenorValorY) {
                    individuoMejor = cont;
                    MenorValorY = valorIndiv;
                }
            }
            return objEcuacion.ValorX(Individuos[individuoMejor]);
        }

        //Genera un individuo booleano al azar
        private string BooleanoAzar(int Tamano) {
            string numero = "";
            for (int cont = 1; cont <= Tamano; cont++) {
                if (azar.NextDouble() < 0.5)
                    numero += "1";
                else
                    numero += "0";
            }
            return numero;
        }
    }
}

```



```

/* Buscar valor de X. Algoritmo genético: Operador cruce */
using System;

namespace Genetico06 {
    class Ecuacion {
        private double minX, maxX;

        //El rango de búsqueda en X para encontrar el mínimo Y
        public void Rango(double minX, double maxX) {
            this.minX = minX;
            this.maxX = maxX;
        }

        //El individuo en binario se convierte a real y
        //retorna el valor que genera la ecuación
        public double ValorY(string Individuo) {
            double x = ValorX(Individuo);
            double y = -1.78 * Math.Pow(Math.Sin(x), 2);
            y += 6.40 * Math.Pow(Math.Cos(x), 2);
            y += 3.07 * Math.Pow(Math.Sin(x), 3);
            return y;
        }

        //Convierte la representación binaria en dato tipo real
        public double ValorX(string Individuo) {
            double multiplica = (maxX - minX);
            multiplica /= (Math.Pow(2, Individuo.Length) - 1);
            int numero = Convert.ToInt32(Individuo, 2);
            return minX + numero * multiplica;
        }
    }
}

```

El resultado obtenido es:

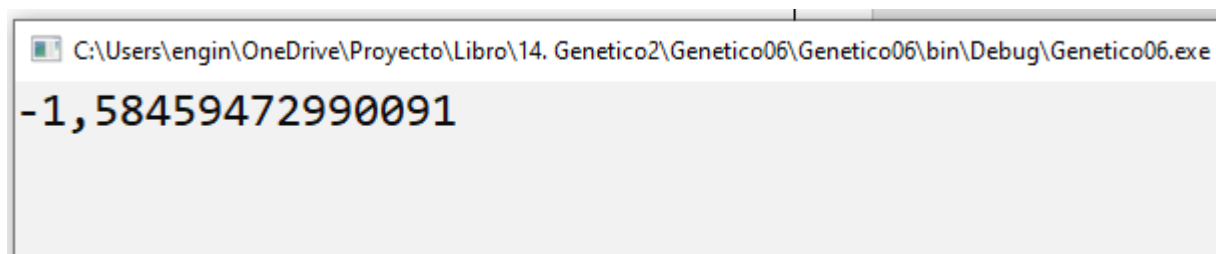


Ilustración 20: Con 100.000 ciclos

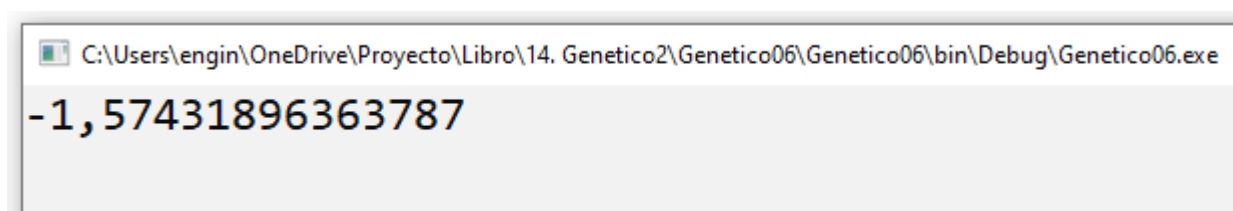


Ilustración 21: Con 150.000 ciclos

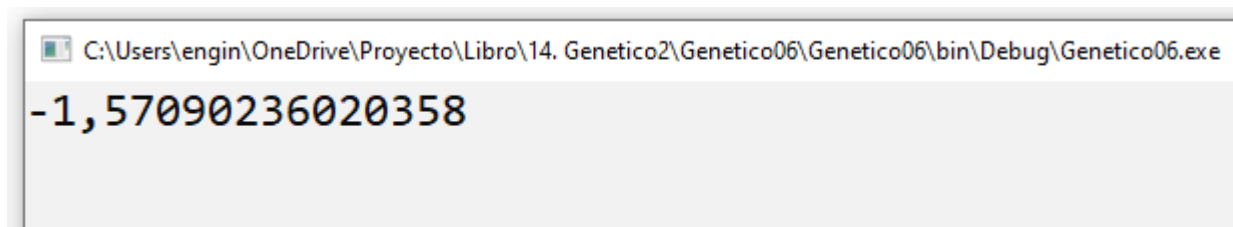


Ilustración 22: Con 200.000 ciclos

Combinando mutación y cruce

Existen varias opciones para combinar los operadores de mutación y cruce.

1. Se hace el cruce y en el hijo resultante se le aplica el operador de mutación.
2. Se hace el cruce, pero no se aplica siempre al hijo resultante el operador de mutación. La mutación se da con un porcentaje de probabilidad.
3. A uno de los padres se le aplica el operador mutación y después se aplica el cruce.

Se hace un cambio a la clase Poblacion, este sería el código en el directorio **Genetico07**:

Program.cs

```
/* Buscar valor de X. Algoritmo genético: Operador cruce y mutación */
using System;

namespace Genetico07 {
    class Program {
        static void Main() {
            double minValorX = -3;
            double maxValorX = 2;
            Ecuacion objEcuacion = new Ecuacion();
            objEcuacion.Rango(minValorX, maxValorX);

            Poblacion objPoblacion = new Poblacion();
            int TamanoIndividuo = 30;
            int TotalIndividuos = 50;
            int TotalCiclos = 100000;
            double MejorIndiv = objPoblacion.Proceso(objEcuacion, TamanoIndividuo, TotalIndividuos, TotalCiclos);

            //Muestra el individuo mejor adaptado
            Console.WriteLine(MejorIndiv);
            Console.ReadKey();
        }
    }
}
```

```

/* Buscar valor de X. Algoritmo genético: Operador cruce y mutación */
using System;
using System.Collections.Generic;

namespace Genetico07 {
    class Poblacion {
        private readonly List<string> Individuos; //Lista de individuos
        private readonly Random azar;

        public Poblacion() {
            Individuos = new List<string>();
            azar = new Random();
        }

        public double Proceso(Ecuacion objEcuacion, int TamanoIndividuo, int numIndividuos, int numCiclos) {

            //Crea la población con individuos generados al azar
            for (int cont = 1; cont <= numIndividuos; cont++) {
                Individuos.Add(BooleanoAzar(TamanoIndividuo));
            }

            //Proceso de algoritmo genético
            for (int ciclo = 1; ciclo <= numCiclos; ciclo++) {

                //Toma dos individuos al azar
                int indivA = azar.Next(Individuos.Count);
                int indivB;
                do {
                    indivB = azar.Next(Individuos.Count);
                } while (indivA == indivB); //Asegura que sean dos individuos distintos

                //Usa el operador cruce
                int posAzar = azar.Next(Individuos[indivA].Length);
                string parteA = Individuos[indivA].Substring(0, posAzar);
                string parteB = Individuos[indivB].Substring(posAzar);
                string HijoA = parteA + parteB;

                //Además muta el hijo
                char[] numeros = HijoA.ToCharArray();
                int pos = azar.Next(HijoA.Length);
                if (numeros[pos] == '0')
                    numeros[pos] = '1';
                else
                    numeros[pos] = '0';
                HijoA = new string(numeros);

                //Evalúa la adaptación de los dos individuos
                double valorIndivA = objEcuacion.ValorY(Individuos[indivA]);
                double valorIndivB = objEcuacion.ValorY(Individuos[indivB]);
                double valorHijoA = objEcuacion.ValorY(HijoA);

                //Si los hijos son mejores que los padres, entonces los reemplaza
                if (valorHijoA < valorIndivA) Individuos[indivA] = HijoA;
                else if (valorHijoA < valorIndivB) Individuos[indivB] = HijoA;
            }

            //Después del ciclo, busca el mejor individuo adaptado de la población
            int individuoMejor = 0;
            double MenorValorY = double.MaxValue;
            for (int cont = 0; cont < Individuos.Count; cont++) {
                double valorIndiv = objEcuacion.ValorY(Individuos[cont]);
                if (valorIndiv < MenorValorY) {
                    individuoMejor = cont;
                    MenorValorY = valorIndiv;
                }
            }
            return objEcuacion.ValorX(Individuos[individuoMejor]);
        }

        //Genera un individuo booleano al azar
        private string BooleanoAzar(int Tamano) {
            string numero = "";
            for (int cont = 1; cont <= Tamano; cont++) {
                if (azar.NextDouble() < 0.5)
                    numero += "1";
                else

```

```

        numero += "0";
    }
    return numero;
}
}
}

```

Ecuacion.cs

```

/* Buscar valor de X. Algoritmo genético: Operador cruce y mutación */
using System;

namespace Genetico07 {
    class Ecuacion {
        private double minX, maxX;

        //El rango de búsqueda en X para encontrar el mínimo Y
        public void Rango(double minX, double maxX) {
            this.minX = minX;
            this.maxX = maxX;
        }

        //El individuo en binario se convierte a real y
        //retorna el valor que genera la ecuación
        public double ValorY(string Individuo) {
            double x = ValorX(Individuo);
            double y = -1.78 * Math.Pow(Math.Sin(x), 2);
            y += 6.40 * Math.Pow(Math.Cos(x), 2);
            y += 3.07 * Math.Pow(Math.Sin(x), 3);
            return y;
        }

        //Convierte la representación binaria en dato tipo real
        public double ValorX(string Individuo) {
            double multiplica = (maxX - minX);
            multiplica /= (Math.Pow(2, Individuo.Length) - 1);
            int numero = Convert.ToInt32(Individuo, 2);
            return minX + numero * multiplica;
        }
    }
}

```

Resultados:

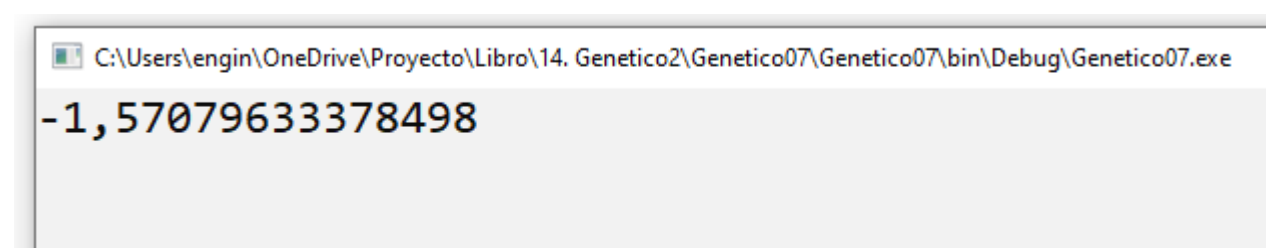


Ilustración 23: Ejecución con 100.000 ciclos

Trazando el camino

Tenemos el siguiente escenario: Un robot está en la posición 0,0 de un arreglo bidimensional de $N \times M$, se desplaza casilla a casilla en dirección Norte, Sur, Este, Oeste y no lo hace en diagonal. En ese arreglo bidimensional hay distribuidas aleatoriamente casillas que son puntos de obligado paso. Se pide entonces como sería la ruta de desplazamiento del robot de tal manera que pase por las casillas de paso obligado. Una ruta de desplazamiento podría ser: "SSSEESSOOOONNE" que significaría desplazarse al Sur tres casillas, luego al Este, dos casillas, de nuevo al Sur, dos casillas, sigue al Oeste, cuatro casillas, al Norte, dos casillas y finalmente al Este, una casilla.

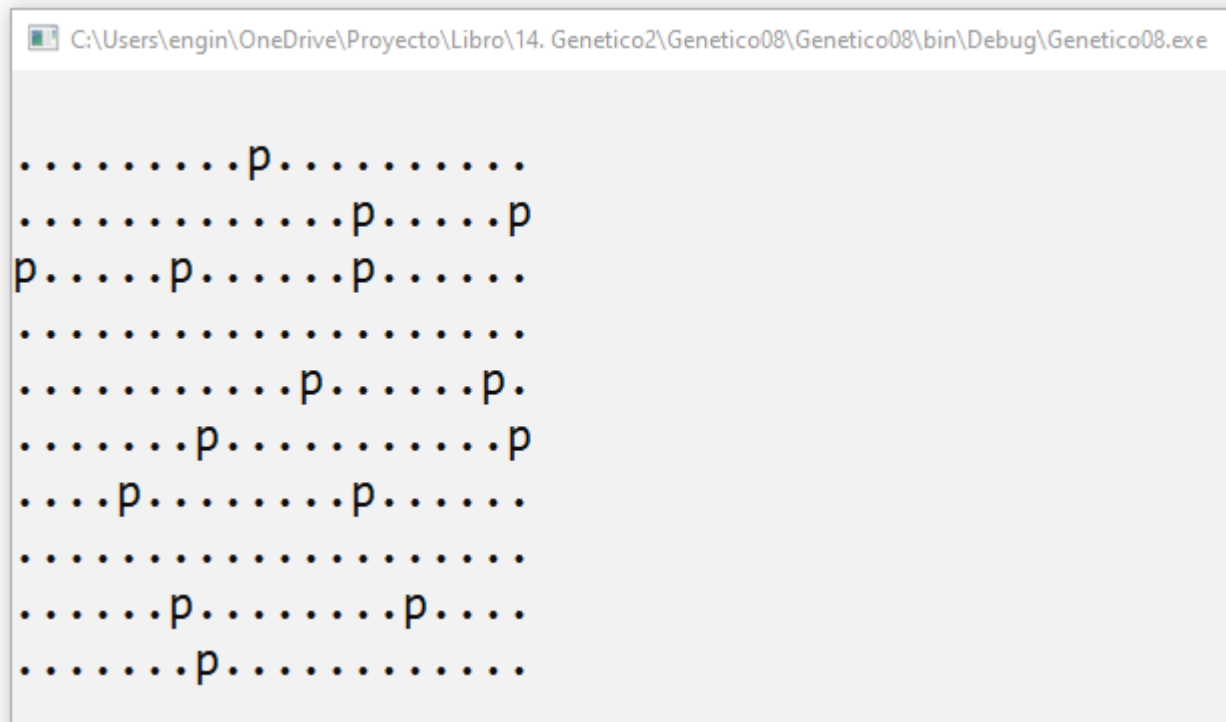


Ilustración 24: Generar una ruta para visitar todos los puntos "p"

Se hace un cambio a la clase Poblacion, este sería el código en el directorio **Genetico08**:

Program.cs

```

/* Buscar la ruta en un tablero para visitar determinados puntos. Uso del operador mutación */
using System;

namespace Genetico08 {
    class Program {
        static void Main() {
            //Genera el tablero con los puntos obligados puestos al azar
            int Filas = 10;
            int Columnas = 20;
            int PuntosObligados = 15;
            Tablero objTablero = new Tablero(Filas, Columnas, PuntosObligados);
            objTablero.ImprimeTabla();

            //Proceso de algoritmo genético
            Poblacion objPoblacion = new Poblacion();
            int TamanoIndividuo = 100;
            int TotalIndividuos = 70;
            int TotalCiclos = 10000;
            string MejorIndiv = objPoblacion.Proceso(objTablero, TamanoIndividuo, TotalIndividuos, TotalCiclos);

            //Muestra el individuo mejor adaptado
            Console.WriteLine(MejorIndiv);
            objTablero.HaceCamino(MejorIndiv);
            objTablero.ImprimeTabla();
            Console.ReadKey();
        }
    }
}

```

```

/* Buscar la ruta en un tablero para visitar determinados puntos. Uso del operador mutación */
using System;
using System.Collections.Generic;

namespace Genetico08 {
    class Poblacion {
        private readonly List<string> Individuos; //Lista de individuos
        private readonly Random azar;

        public Poblacion() {
            Individuos = new List<string>();
            azar = new Random();
        }

        public string Proceso(Tablero objTablero, int TamanoIndividuo, int numIndividuos, int numCiclos) {

            //Crea la población con individuos generados al azar
            for (int cont = 1; cont <= numIndividuos; cont++) {
                Individuos.Add(RutaAzar(TamanoIndividuo));
            }

            //Proceso de algoritmo genético
            for (int ciclo = 1; ciclo <= numCiclos; ciclo++) {

                //Toma dos individuos al azar
                int indivA = azar.Next(Individuos.Count);
                int indivB;
                do {
                    indivB = azar.Next(Individuos.Count);
                } while (indivA == indivB); //Asegura que sean dos individuos distintos

                //Evalúa la adaptación de los dos individuos
                double valorIndivA = objTablero.EvaluaRuta(Individuos[indivA]);
                double valorIndivB = objTablero.EvaluaRuta(Individuos[indivB]);

                //El mejor individuo reemplaza al peor y la copia se muta
                if (valorIndivA < valorIndivB) {
                    Individuos[indivB] = Individuos[indivA];
                    Muta(indivB);
                }
                else {
                    Individuos[indivA] = Individuos[indivB];
                    Muta(indivA);
                }
            }

            //Después del ciclo, busca el mejor individuo adaptado de la población
            int individuoMejor = 0;
            int MenorValorY = 99999;
            for (int cont = 0; cont < Individuos.Count; cont++) {
                int valorIndiv = objTablero.EvaluaRuta(Individuos[cont]);
                if (valorIndiv < MenorValorY) {
                    individuoMejor = cont;
                    MenorValorY = valorIndiv;
                }
            }
            return Individuos[individuoMejor];
        }

        //Genera un individuo ruta al azar
        private string RutaAzar(int Tamano) {
            string ruta = "";
            string permite = "SE";
            for (int cont = 1; cont <= Tamano; cont++) {
                int pos = azar.Next(permite.Length);
                ruta += permite[pos];
                switch (permite[pos]) {
                    case 'N': permite = "NEO"; break;
                    case 'S': permite = "SEO"; break;
                    case 'E': permite = "ENS"; break;
                    case 'O': permite = "ONS"; break;
                }
            }
            return ruta;
        }
    }
}

```

```
//Muta individuos
private void Muta(int individuo) {
    string permite = "";
    char[] cambia = Individuos[individuo].ToCharArray();
    int posAzar = azar.Next(Individuos[individuo].Length);
    switch (cambia[posAzar]) {
        case 'N': permite = "SEO"; break;
        case 'S': permite = "NEO"; break;
        case 'E': permite = "NSO"; break;
        case 'O': permite = "NSE"; break;
    }
    int nuevo = azar.Next(permite.Length);
    cambia[posAzar] = permite[nuevo];
    Individuos[individuo] = new string(cambia);
}
}
```

```

/* Buscar la ruta en un tablero para visitar determinados puntos. Uso del operador mutación */
using System;

namespace Genetico08 {
    class Tablero {
        private int Filas, Columnas;
        private char[,] tabla;

        //Crea el tablero con los puntos
        public Tablero(int Filas, int Columnas, int Puntos) {
            this.Filas = Filas;
            this.Columnas = Columnas;

            //Tablero lleno con 0s
            int fila, columna;
            tabla = new char[Filas, Columnas];
            for (fila = 0; fila < Filas; fila++)
                for (columna = 0; columna < Columnas; columna++)
                    tabla[fila, columna] = '.';

            //Pone los puntos de paso obligado al azar
            Random azar = new Random();
            for (int punto = 1; punto <= Puntos; punto++) {
                do {
                    fila = azar.Next(Filas);
                    columna = azar.Next(Columnas);
                } while (tabla[fila, columna] == 'p');
                tabla[fila, columna] = 'p';
            }
        }

        //Imprime la tabla
        public void ImprimeTabla() {
            for (int fila = 0; fila < Filas; fila++) {
                Console.WriteLine(" ");
                for (int columna = 0; columna < Columnas; columna++)
                    Console.Write(tabla[fila, columna]);
            }
            Console.WriteLine(" ");
            Console.WriteLine(" ");
        }

        //Evalúa la ruta que hace el individuo para ver si visita los puntos obligados
        //Retorna cuantos puntos le queda por visitar. Si es cero entonces ha visitado todos.
        public int EvaluaRuta(string ruta) {
            HacerCamino(ruta);

            //Evalua la efectividad de la ruta.
            int SinVisitar = 0;
            for (int fila = 0; fila < Filas; fila++) {
                for (int columna = 0; columna < Columnas; columna++)
                    if (tabla[fila, columna] == 'p')
                        SinVisitar++;
            }

            RestaurarTabla();

            return SinVisitar; //Si vale cero, entonces ha visitado todos los puntos
        }

        //Trazar el camino
        public void HacerCamino(string ruta) {
            //Inicia siempre en 0,0
            int posF = 0;
            int posC = 0;

            //Pone la ruta en el tablero
            for (int letra = 0; letra < ruta.Length; letra++) {
                switch (ruta[letra]) {
                    case 'N': if (posF > 0) posF--; break;
                    case 'S': if (posF < this.Filas - 1) posF++; break;
                    case 'E': if (posC < this.Columnas - 1) posC++; break;
                    case 'O': if (posC > 0 ) posC--; break;
                }

                if (tabla[posF, posC] == '.')

```



```
        tabla[posF, posC] = 'x';
    else if (tabla[posF, posC] == 'p')
        tabla[posF, posC] = 'V';
    }
}

//Restaura la tabla a sólo puntos y puntos de obligado paso
private void RestaurarTabla() {
    for (int fila = 0; fila < Filas; fila++) {
        for (int columna = 0; columna < Columnas; columna++) {
            if (tabla[fila, columna] == 'x') tabla[fila, columna] = '.';
            if (tabla[fila, columna] == 'V') tabla[fila, columna] = 'p';
        }
    }
}
}
```

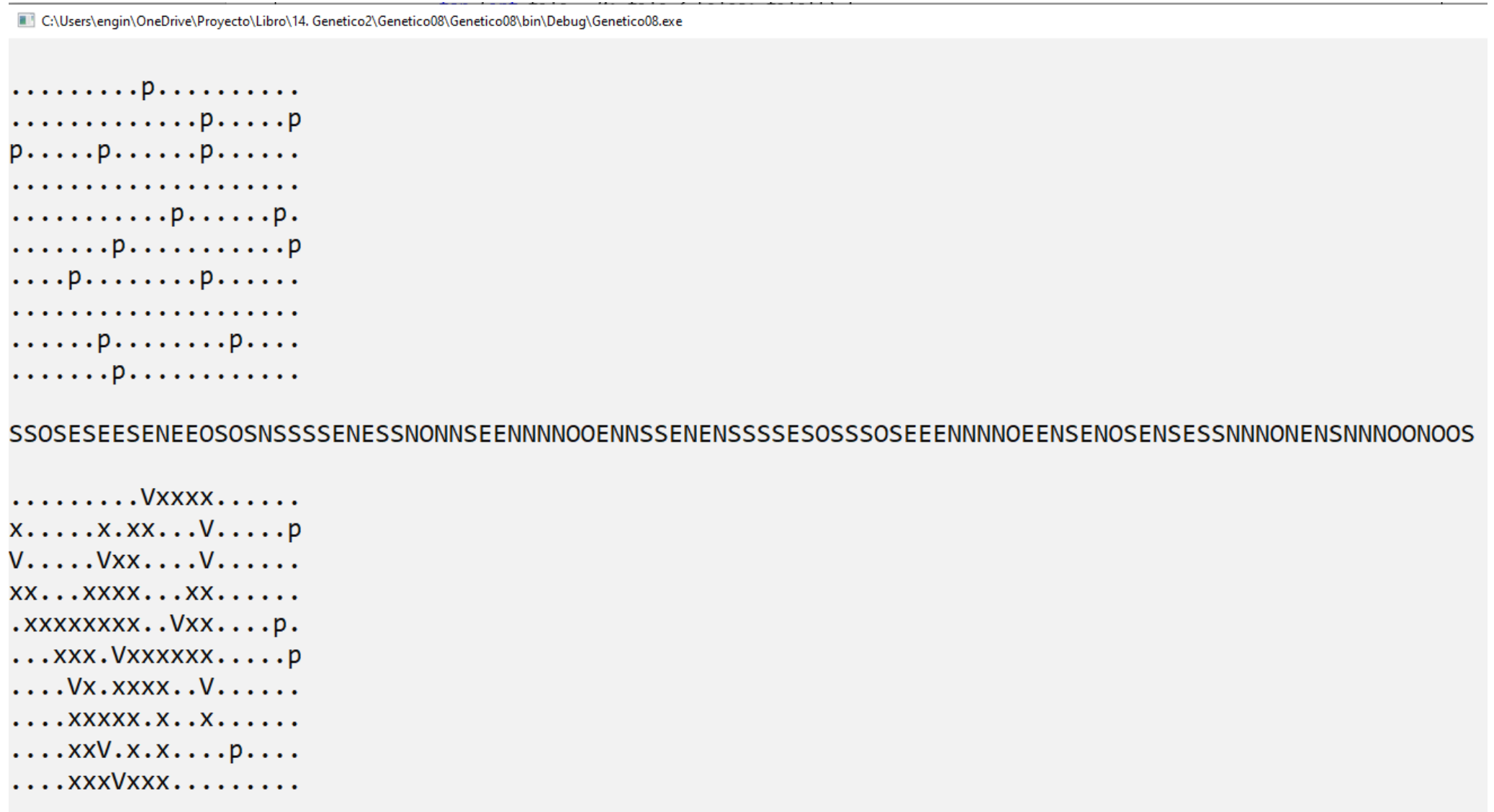


Ilustración 25: Mejor ruta encontrada, los "V" son los puntos obligados visitados, los "p" fueron los puntos obligado que no se visitó

El algoritmo se aproxima dar con la cadena que es la ruta del robot para visitar los puntos obligados. Sin embargo, no siempre los logra visitar todos, el número de 10000 ciclos o el tamaño de la cadena que es de 100 puede quedarse cortos. Además, las instrucciones de ruta a veces se anulan entre sí, por ejemplo, “SN” significa ve al Sur y luego al Norte, por lo que vuelve al inicio.

Una consideración sobre este algoritmo

Un problema detectado con este algoritmo es que, si se muta cerca al inicio, el impacto es mucho más grande. Por ejemplo, si se modifica el primer paso y en vez de ir al Sur, se modifica que vaya al Este, cambiaría totalmente la ruta de allí en adelante, es decir, un “efecto mariposa”.

Juego de aritmética

Tenemos el siguiente tablero:

2	-	5	+	9	=	6
+		-		+		
3	+	1	-	8	=	-4
*		*		-		
4	*	6	+	7	=	31
=		=		=		
14		-1		10		

Tanto en horizontal como en vertical hay operaciones matemáticas, respetando las reglas algebraicas como la precedencia de las operaciones (primero multiplicaciones, luego sumas y restas).

Ahora nos dan el mismo tablero, pero borrando los números que entran en las operaciones

A	-	B	+	C	=	6
+		-		+		
D	+	E	-	F	=	-4
*		*		-		
G	*	H	+	I	=	31
=		=		=		
14		-1		10		

Las ecuaciones entonces serían:

$A - B + C = 6$

$D + E - F = -4$

$G * H + I = 31$

$A + D * G = 14$

$B - E * H = -1$

$C + F - I = 10$

El juego consiste en dar con los valores de esas variables (A, B, C, D, E, F, G, H, I) para que den con esos resultados.

Un algoritmo genético puede dar una muy buena aproximación:

- 1. Generar una población donde cada individuo tiene 9 números enteros entre 0 y 20
- 2. Se seleccionan dos individuos al azar de esa población y se evalúan.
- 3. La evaluación de cada individuo es comparar sus resultados en las operaciones en horizontal como en vertical con los resultados esperados. Esa comparación es la diferencia absoluta entre cada operación horizontal y vertical del número y el resultado esperado. Se suman todas las diferencias absolutas y aquel individuo que se acerca más a cero es el ganador.
- 4. El individuo ganador sobre-escribe sus números sobre el individuo perdedor.
- 5. Se modifica la copia en un número.

Para las pruebas, el software genera un tablero con números y operaciones al azar y el algoritmo genético busca dar con esos números.

El código en C# en el directorio **Genetico09**:

Program.cs

```
/* Juego de aritmética. Uso del operador mutación */
using System;

namespace Genetico09 {
    class Program {
        static void Main() {
            /*    Un ejemplo de juego de aritmética
            *    2 - 5 + 9 = 6
            *    3 + 1 - 8 = -4
            *    4 * 6 + 7 = 31
            *    2 + 3 * 4 = 14
            *    5 - 1 * 6 = -1
            *    9 + 8 - 7 = 10
            *
            *    Se pone en el algoritmo genético
            *    ¿Podrá deducir los mismos números?
            * */
            string[] operaciones = { "A-B+C=6",
                                     "D+E-F=-4",
                                     "G*H+I=31",
                                     "A+D*G=14",
                                     "B-E*H=-1",
                                     "C+F-I=10" };

            //Proceso de algoritmo genético
            Poblacion objPoblacion = new Poblacion();
            int TotalIndividuos = 80;
            int TotalCiclos = 500000;
            int[] MejorIndiv = objPoblacion.Proceso(operaciones, TotalIndividuos, TotalCiclos);

            //Imprime las operaciones
            for (int cont = 0; cont < 6; cont++)
                Console.WriteLine(operaciones[cont]);
            Console.WriteLine(" ");

            //Imprime los valores
            int varA=0, varB=0, varC=0;
            for (int cont = 0; cont < 6; cont++) {
                switch (cont) {
                    case 0:
                        varA = MejorIndiv[0];
                        varB = MejorIndiv[1];
                        varC = MejorIndiv[2];
                        break;
                    case 1:
                        varA = MejorIndiv[3];
                        varB = MejorIndiv[4];
                        varC = MejorIndiv[5];
                        break;
                    case 2:
                        varA = MejorIndiv[6];
                        varB = MejorIndiv[7];
                        varC = MejorIndiv[8];
                        break;
                    case 3:
                        varA = MejorIndiv[0];
                        varB = MejorIndiv[3];
                        varC = MejorIndiv[6];
                        break;
                    case 4:
                        varA = MejorIndiv[1];
                        varB = MejorIndiv[4];
                        varC = MejorIndiv[7];
                        break;
                    case 5:
                        varA = MejorIndiv[2];
                        varB = MejorIndiv[5];
                        varC = MejorIndiv[8];
                        break;
                }

                char opA = operaciones[cont][1];
                char opB = operaciones[cont][3];
            }
        }
    }
}
```

```

        int resultado = objPoblacion.RetornaOperacion(varA, opA, varB, opB, varC);

        Console.Write(varA.ToString() + " ");
        Console.Write(opA.ToString() + " ");
        Console.Write(varB.ToString() + " ");
        Console.Write(opB.ToString() + " ");
        Console.Write(varC.ToString());
        Console.WriteLine(" = " + resultado.ToString());
    }

    Console.ReadKey();
}
}
}

```

Poblacion.cs

```

/* Juego de aritmética. Uso del operador mutación */
using System;
using System.Collections.Generic;

namespace Genetico09 {
    class Poblacion {
        private List<int[]> Individuos; //Lista de individuos
        private Random azar;

        public Poblacion() {
            Individuos = new List<int[]>();
            azar = new Random();
        }

        public int[] Proceso(string[] operaciones, int numIndividuos, int numCiclos) {
            //Acelera la evaluación al convertir a números los resultados de las operaciones
            int[] resultados = new int[6];
            for (int cont = 0; cont < 6; cont++) {
                string numero = operaciones[cont].Substring(6);
                resultados[cont] = Convert.ToInt32(numero);
            }

            //Crea la población con individuos (las 9 variables del juego de matemática) generados al azar
            for (int cont = 1; cont <= numIndividuos; cont++) {
                int[] valor = new int[9];
                for (int num = 0; num < 9; num++)
                    valor[num] = azar.Next(0, 20);
                Individuos.Add(valor);
            }

            //Proceso de algoritmo genético
            for (int ciclo = 1; ciclo <= numCiclos; ciclo++) {

                //Toma dos individuos al azar
                int indivA = azar.Next(Individuos.Count);
                int indivB;
                do {
                    indivB = azar.Next(Individuos.Count);
                } while (indivA == indivB); //Asegura que sean dos individuos distintos

                //Evalúa la adaptación de los dos individuos
                double valorIndivA = EvaluaIndividuo(Individuos[indivA], operaciones, resultados);
                double valorIndivB = EvaluaIndividuo(Individuos[indivB], operaciones, resultados);

                //El mejor individuo reemplaza al peor y la copia se muta
                if (valorIndivA < valorIndivB) {
                    CopiaIndividuo(indivA, indivB);
                    Muta(indivB);
                }
                else {
                    CopiaIndividuo(indivB, indivA);
                    Muta(indivA);
                }
            }

            //Después del ciclo, busca el mejor individuo adaptado de la población
            int individuoMejor = 0;
            double MenorValorY = Double.MaxValue;
            for (int cont = 0; cont < Individuos.Count; cont++) {
                double valorIndiv = EvaluaIndividuo(Individuos[cont], operaciones, resultados);
            }
        }
    }
}

```

```

        if (valorIndiv < MenorValorY) {
            individuoMejor = cont;
            MenorValorY = valorIndiv;
        }
    }
    return Individuos[individuoMejor];
}

//Copia el individuo número a número
private void CopiaIndividuo(int origen, int destino) {
    for (int cont = 0; cont < 6; cont++)
        Individuos[destino][cont] = Individuos[origen][cont];
}

//Evalúa el individuo si se acerca a los resultados esperados
private int EvaluaIndividuo(int[] Variables, string[] operaciones, int[] resultados) {
    int diferencia = 0;
    int varA=0, varB=0, varC=0;

    for (int cont=0; cont < 6; cont++) {
        //Las 9 variables
        switch (cont) {
            case 0:
                varA = Variables[0];
                varB = Variables[1];
                varC = Variables[2];
                break;
            case 1:
                varA = Variables[3];
                varB = Variables[4];
                varC = Variables[5];
                break;
            case 2:
                varA = Variables[6];
                varB = Variables[7];
                varC = Variables[8];
                break;
            case 3:
                varA = Variables[0];
                varB = Variables[3];
                varC = Variables[6];
                break;
            case 4:
                varA = Variables[1];
                varB = Variables[4];
                varC = Variables[7];
                break;
            case 5:
                varA = Variables[2];
                varB = Variables[5];
                varC = Variables[8];
                break;
        }
        //Trae las operaciones
        char opA = operaciones[cont][1];
        char opB = operaciones[cont][3];

        //Hace la operación y compara con el resultado esperado
        int resultado = RetornaOperacion(varA, opA, varB, opB, varC);
        diferencia += Math.Abs(resultados[cont] - resultado);
    }
    return diferencia;
}

//Hace las operaciones y retorna el resultado de estas
public int RetornaOperacion(int varA, char opA, int varB, char opB, int varC) {
    if (opA == '+' && opB == '+') return varA + varB + varC;
    if (opA == '+' && opB == '-') return varA + varB - varC;
    if (opA == '+' && opB == '*') return varA + varB * varC;

    if (opA == '-' && opB == '+') return varA - varB + varC;
    if (opA == '-' && opB == '-') return varA - varB - varC;
    if (opA == '-' && opB == '*') return varA - varB * varC;

    if (opA == '*' && opB == '+') return varA * varB + varC;
    if (opA == '*' && opB == '-') return varA * varB - varC;
    if (opA == '*' && opB == '*') return varA * varB * varC;
}

```

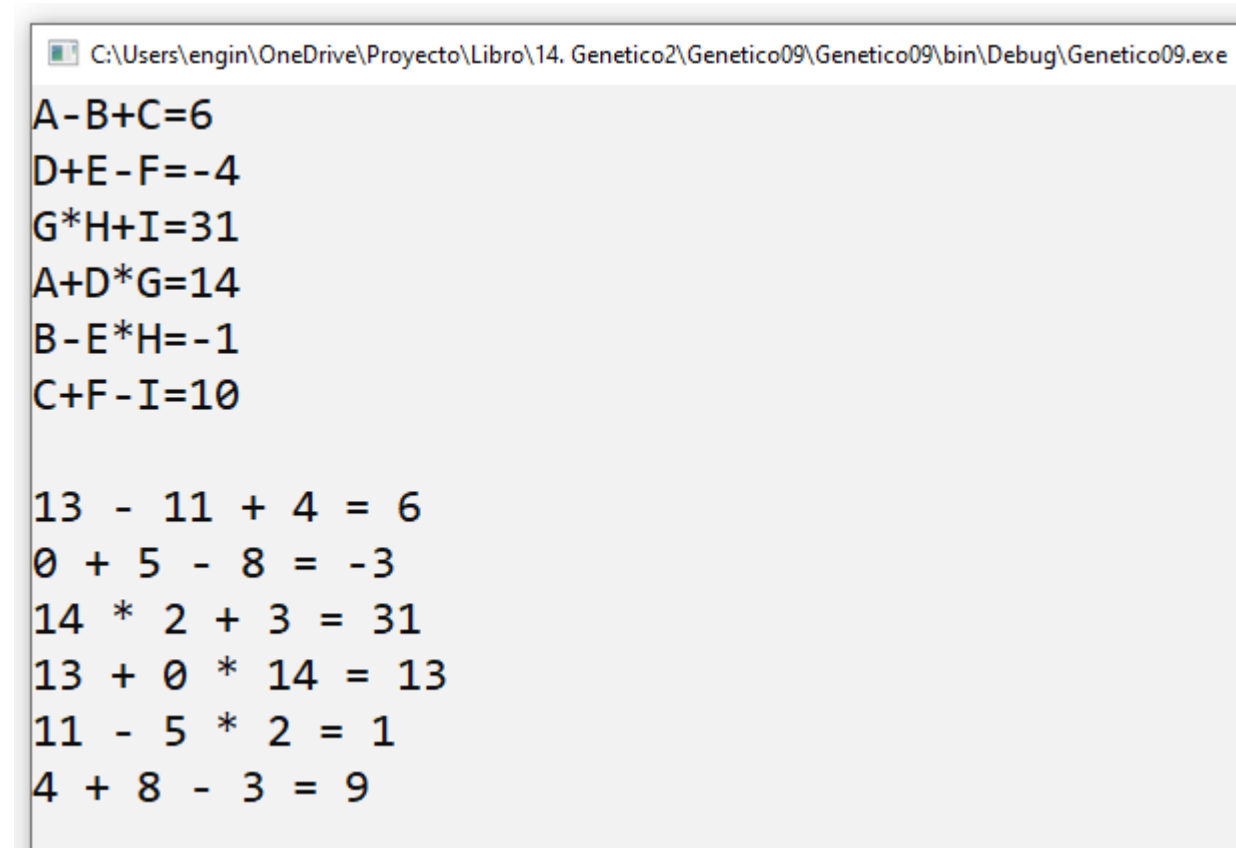
```

        return 0;
    }

    //Muta el individuo generando un número entero entre 0 y 19
    private void Muta(int indiv) {
        int pos = azar.Next(7);
        Individuos[indiv][pos] = azar.Next(0, 20);
    }
}
}

```

Ejemplos de ejecución:



```

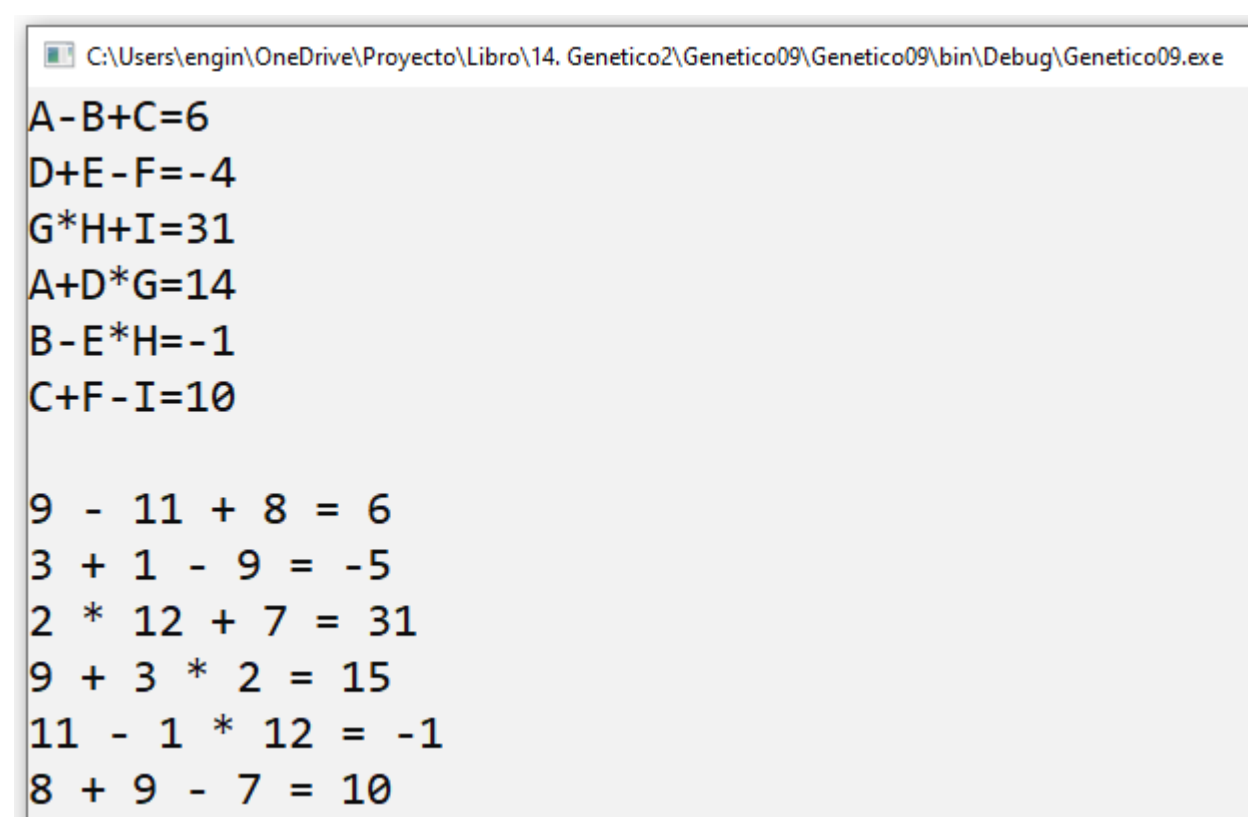
C:\Users\engin\OneDrive\Proyecto\Libro\14. Genetico2\Genetico09\Genetico09\bin\Debug\Genetico09.exe
A-B+C=6
D+E-F=-4
G*H+I=31
A+D*G=14
B-E*H=-1
C+F-I=10

13 - 11 + 4 = 6
0 + 5 - 8 = -3
14 * 2 + 3 = 31
13 + 0 * 14 = 13
11 - 5 * 2 = 1
4 + 8 - 3 = 9

```

Ilustración 26: Tablero generado con 100.000 ciclos y 1000 individuos

El código mostrado hace uso de una población de 1000 individuos y el ciclo tiene 100.000 iteraciones. Si comparamos el tablero generado (que sería el que se debería hallar) con el tablero hallado (que es lo que concluye por el algoritmo genético), se nota que la aproximación es buena tanto en operaciones horizontales como verticales. Como son 6 ecuaciones y 9 variables, no hay respuesta única.



```

C:\Users\engin\OneDrive\Proyecto\Libro\14. Genetico2\Genetico09\Genetico09\bin\Debug\Genetico09.exe
A-B+C=6
D+E-F=-4
G*H+I=31
A+D*G=14
B-E*H=-1
C+F-I=10

9 - 11 + 8 = 6
3 + 1 - 9 = -5
2 * 12 + 7 = 31
9 + 3 * 2 = 15
11 - 1 * 12 = -1
8 + 9 - 7 = 10

```

Ilustración 27: Tablero generado con 1 millón de ciclos y 1000 individuos

El problema de las Reinas

Dado un tablero de ajedrez estándar de 8*8, se deben ubicar 8 reinas sin que ninguna amenace a otra. Cabe recordar que la Reina es la pieza más poderosa del ajedrez con ataques en horizontal y diagonal.

Se ha dejado flexible el programa para escalar a un tablero de N*N, sólo es cambiar la línea:

```
int tamTablero = 8; //Tamaño del tablero cuadrado a ubicar las reinas
```

El reto, entonces, es poner en un tablero de N*N, exactamente N reinas sin que se maten entre sí. Para eso, cada individuo es un arreglo unidimensional de tamaño N, donde en cada posición se tendría almacenado en que fila estaría la reina ubicada.

Ejemplo:

7	0	4	2	5	3	6	1
---	---	---	---	---	---	---	---

Significa que en la columna 0, la primera Reina se ubicará en la fila 7, en la columna 1, la segunda Reina se ubicará en la fila 0, la tercera Reina se ubicará en la fila 4 y así sucesivamente.

En la generación de los individuos, se programa para que no se repita ninguna fila. De esa manera, al menos, el programa garantiza que no se maten horizontalmente, ni verticalmente.

El código en C# a continuación está en el directorio **Genetico10**:

Program.cs

```
//Algoritmo genético para ubicar las reinas de un ajedrez sin que se ataquen entre sí. Operador mutación
using System;

namespace Genetico10 {
    class Program {
        static void Main() {
            int tamTablero = 8; //Tamaño del tablero cuadrado a ubicar las reinas

            //Proceso de algoritmo genético
            Poblacion objPoblacion = new Poblacion(tamTablero);
            int TotalIndividuos = 30;
            int TotalCiclos = 10000;
            int[] Reinas = objPoblacion.Proceso(TotalIndividuos, TotalCiclos);

            objPoblacion.ImprimeTablero(Reinas);
            Console.ReadKey();
        }
    }
}
```

```

//Algoritmo genético para ubicar las reinas de un ajedrez sin que se ataquen entre sí. Operador mutación
using System;
using System.Collections.Generic;

namespace Genetico10 {
    class Poblacion {
        private List<int[]> Reinas; //Lista de conjuntos de reinas
        private Random azar;
        private char[,] Tablero;

        public Poblacion(int tamTablero) {
            Reinas = new List<int[]>();
            azar = new Random();
            Tablero = new char[tamTablero, tamTablero];
        }

        public int[] Proceso(int numIndividuos, int numCiclos) {
            //Crea la población con individuos (reinas) generados al azar
            //En un principio las posiciones de las reinas sería [0,1,2,3,4,5,6,7,8,..., tamTablero-1]
            for (int cont = 1; cont <= numIndividuos; cont++) {
                int[] valor = new int[Tablero.GetLength(0)];
                for (int num = 0; num < Tablero.GetLength(0); num++) valor[num] = num;
                Reinas.Add(valor);
            }

            //Proceso de algoritmo genético
            for (int ciclo = 1; ciclo <= numCiclos; ciclo++) {

                //Toma dos conjuntos de reinas al azar
                int conjuntoA = azar.Next(Reinas.Count);
                int conjuntoB;
                do {
                    conjuntoB = azar.Next(Reinas.Count);
                } while (conjuntoA == conjuntoB); //Asegura que sean dos conjuntos distintos

                //Evalúa la adaptación de los dos conjuntos
                int valorConjuntoA = EvaluaReinas(conjuntoA);
                int valorConjuntoB = EvaluaReinas(conjuntoB);

                //El mejor conjunto reemplaza al peor y la copia se muta
                if (valorConjuntoA > valorConjuntoB) {
                    CopiaConjunto(conjuntoA, conjuntoB);
                    Muta(conjuntoB);
                }
                else {
                    CopiaConjunto(conjuntoB, conjuntoA);
                    Muta(conjuntoA);
                }
            }

            //Después del ciclo, busca el mejor conjunto de reinas
            int ConjuntoMejor = 0;
            int MayorNumReinas = -1;
            for (int cont = 0; cont < Reinas.Count; cont++) {
                int valorConjunto = EvaluaReinas(cont);
                if (valorConjunto > MayorNumReinas) {
                    ConjuntoMejor = cont;
                    MayorNumReinas = valorConjunto;
                }
            }

            return Reinas[ConjuntoMejor];
        }

        private void CopiaConjunto(int origen, int destino) {
            for (int cont = 0; cont < Tablero.GetLength(0); cont++)
                Reinas[destino][cont] = Reinas[origen][cont];
        }

        private void Muta(int indiv) {
            int posA = azar.Next(Tablero.GetLength(0));
            int posB;
            do {
                posB = azar.Next(Tablero.GetLength(0));
            } while (posA == posB); //Asegura que sean dos posiciones distintas
        }
    }
}

```



```

//Intercambia posiciones de la reina
int temp = Reinas[indiv][posA];
Reinas[indiv][posA] = Reinas[indiv][posB];
Reinas[indiv][posB] = temp;
}

public int EvaluaReinas(int conjunto) {
    IniciaTablero();

    //Ataque de las reinas
    for(int columna=0; columna < Tablero.GetLength(0); columna++) {
        int fila = Reinas[conjunto][columna];
        AtaqueReina(fila, columna);
    }

    //Cuenta el número de R que quedan en el tablero
    int numeroR = 0;
    for(int fila=0; fila < Tablero.GetLength(0); fila++)
        for(int columna=0; columna < Tablero.GetLength(0); columna++) {
            if (Tablero[fila, columna] == 'R') numeroR++;
        }

    return numeroR;
}

//Inicializa el tablero
public void IniciaTablero() {
    for (int fila = 0; fila < Tablero.GetLength(0); fila++)
        for (int columna = 0; columna < Tablero.GetLength(0); columna++)
            Tablero[fila, columna] = '.';
}

//Pone x en el ataque de la Reina
public void AtaqueReina(int fila, int columna) {
    int ataque = 0;
    while (fila - ataque >= 0 && columna - ataque >= 0) {
        Tablero[fila - ataque, columna - ataque] = 'x';
        ataque++;
    }

    ataque = 0;
    while (fila + ataque < Tablero.GetLength(0) && columna + ataque < Tablero.GetLength(0)) {
        Tablero[fila + ataque, columna + ataque] = 'x';
        ataque++;
    }

    ataque = 0;
    while (fila - ataque >= 0 && columna + ataque < Tablero.GetLength(0)) {
        Tablero[fila - ataque, columna + ataque] = 'x';
        ataque++;
    }

    ataque = 0;
    while (fila + ataque < Tablero.GetLength(0) && columna - ataque >= 0) {
        Tablero[fila + ataque, columna - ataque] = 'x';
        ataque++;
    }

    ataque = 0;
    while (fila + ataque < Tablero.GetLength(0)) {
        Tablero[fila + ataque, columna] = 'x';
        ataque++;
    }

    ataque = 0;
    while (fila - ataque >= 0) {
        Tablero[fila - ataque, columna] = 'x';
        ataque++;
    }

    ataque = 0;
    while (columna + ataque < Tablero.GetLength(0)) {
        Tablero[fila, columna + ataque] = 'x';
        ataque++;
    }

    ataque = 0;
    while (columna - ataque >= 0) {

```

```

        Tablero[fila, columna - ataque] = 'x';
        ataque++;
    }

    Tablero[fila, columna] = 'R';
}

public void ImprimeTablero(int[] Reinas) {
    /* Imprime el conjunto */
    Console.Write("Ubicación reinas: ");
    for (int num = 0; num < Reinas.Length; num++)
        Console.Write(Reinas[num].ToString() + ", ");
    Console.WriteLine(" ");

    IniciaTablero();

    //Ataque de las reinas
    for (int columna = 0; columna < Tablero.GetLength(0); columna++) {
        int fila = Reinas[columna];
        AtaqueReina(fila, columna);
    }

    for (int fila = 0; fila < Tablero.GetLength(0); fila++) {
        Console.WriteLine(" ");
        for (int columna = 0; columna < Tablero.GetLength(0); columna++)
            Console.Write(Tablero[fila, columna]);
    }
}
}
}
}

```

Ejemplo de ejecución con un tablero de 8*8

Ilustración 28: Tablero de 8*8.

Ilustración 29: Tablero de 8*8

```
C:\Users\engin\OneDrive\Proyecto\Libro\14. Genetico2\Genetico10\Genetico10\bin\Debug\Genetico10.exe
Ubicación reinas: 0, 6, 3, 5, 7, 1, 4, 2,

Rxxxxxxx
xxxxxRxx
xxxxxxxR
xxRxxxxx
xxxxxxRx
xxxRxxxx
xRxxxxxx
xxxxRxxx
```

Ilustración 30: Tablero de 8*8

```
C:\Users\engin\OneDrive\Proyecto\Libro\14. Genetico2\Genetico10\Genetico10\bin\Debug\Genetico10.exe
Ubicación reinas: 7, 4, 2, 0, 6, 11, 14, 8, 13, 5, 3, 1, 10, 12, 9,

xxxRxxxxxxxxxxx
xxxxxxxxxxxxRxxx
xxRxxxxxxxxxxxxx
xxxxxxxxxxxxRxxxx
xRxxxxxxxxxxxxxx
xxxxxxxxxxRxxxxx
xxxxRxxxxxxxxxxx
Rxxxxxxxxxxxxxxx
xxxxxxxRxxxxxxx
xxxxxxxxxxxxxxxR
xxxxxxxxxxxxxxxRxx
xxxxxRxxxxxxxxxx
xxxxxxxxxxxxxxxRx
xxxxxxxxxxRxxxxxx
xxxxxxRxxxxxxxxx
```

Ilustración 31: Tablero de 15*15

Creando Sudokus

El Sudoku es un juego de mesa con condiciones muy sencillas. Es un tablero de 9*9, en cada fila se ponen los números del 1 al 9 sin repetirse, sucede igual en las columnas, números del 1 al 9 sin repetirse. Adicional a eso, cada grupo de 3*3 en el tablero tiene los números del 1 al 9 sin repetirse. Ver a continuación:

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

El reto es generar diversos tableros Sudokus que cumplan con las condiciones.

Factor de evaluación. Clave para los algoritmos genéticos.

Es muy importante hacer saber al algoritmo genético que la búsqueda de la solución va por buen camino o, al contrario, se está alejando de la solución. Un ejemplo de **mala** evaluación es poner un booleano donde "true" sea que el Sudoku es completamente correcto y "false" si tiene alguna falla. En ese sentido, no habría diferencia entre un tablero que esté vacío o completamente lleno con un mismo número (el peor de los casos) con otro sudoku que le falta muy poco para ser correcto. Luego sería una espera indefinida para el algoritmo genético, porque un tablero generado completamente al azar y que no cumpla, tendría el mismo peso que un tablero que le falta muy poco para completarse.

La solución entonces es evaluar en detalle los tableros y darle el mayor puntaje cuando el sudoku está terminado y el menor puntaje si se da el peor caso de tal manera que el rango sea amplio y se sepa cuando hay progreso o no.

Se evalúa entonces fila por fila buscando que ningún número se repita, la no repetición del número se le da un puntaje de uno(1), luego una fila con ningún número repetido tendría máximo nueve(9) puntos, sucede lo mismo con las columnas y con los grupos de 3*3. Estos serían los casos:

Peor caso

8	8	8	8	8	8	8	8	8	0
8	8	8	8	8	8	8	8	8	0
8	8	8	8	8	8	8	8	8	0
8	8	8	8	8	8	8	8	8	0
8	8	8	8	8	8	8	8	8	0
8	8	8	8	8	8	8	8	8	0
8	8	8	8	8	8	8	8	8	0
8	8	8	8	8	8	8	8	8	0
8	8	8	8	8	8	8	8	8	0
0	0	0	0	0	0	0	0	0	0

$$Puntaje_{Total} = \sum_{fila=1}^9 Puntaje(fila) + \sum_{columna=1}^9 Puntaje(columna) + \sum_{cuadro=1}^9 Puntaje(cuadro)$$

$$Puntaje_{Total} = 0 + 0 + 0 = 0$$

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

$$Puntaje_{Total} = \sum_{fila=1}^9 Puntaje(fila) + \sum_{columna=1}^9 Puntaje(columna) + \sum_{cuadro=1}^9 Puntaje(cuadro)$$

$$Puntaje_{Total} = 81 + 81 + 81 = 243$$

El rango está entre 0 y 243, donde 0 es el peor caso y 243 el mejor caso (Sudoku terminado).

Este sería el código en C# en el directorio **Genetico11**

Program.cs

```
//Algoritmo genético para generar Sudokus completos. Operador mutación
using System;

namespace Genetico11 {
    class Program {
        static void Main() {
            //Proceso de algoritmo genético
            Poblacion objPoblacion = new Poblacion();
            int TotalIndividuos = 50;
            int[] individuo = objPoblacion.Proceso(TotalIndividuos);

            objPoblacion.ImprimeSudoku(individuo);
            Console.ReadKey();
        }
    }
}
```

Poblacion.cs

```
//Algoritmo genético para generar Sudokus completos. Operador mutación
using System;
using System.Collections.Generic;

namespace Genetico11 {
    class Poblacion {
        private List<int[]> Individuo;
        private Random azar;

        public Poblacion() {
            Individuo = new List<int[]>();
            azar = new Random();
        }

        public int[] Proceso(int numIndividuos) {

            //Crea la población con individuos generados al azar
            for (int cont = 1; cont <= numIndividuos; cont++) {
                int[] valor = new int[81];
                for (int num = 0; num < valor.Length; num++) {
                    valor[num] = azar.Next(1, 10);
                }
            }
        }
    }
}
```

```

        Individuo.Add(valor);
    }

    //Proceso de algoritmo genético
    ValidaSudoku objValida = new ValidaSudoku();
    while (true) {

        //Toma dos conjuntos de individuos al azar
        int indivA = azar.Next(Individuo.Count);
        int indivB;
        do {
            indivB = azar.Next(Individuo.Count);
        } while (indivA == indivB); //Asegura que sean dos conjuntos distintos

        //Evalúa la adaptación de los dos conjuntos
        int valorIndividuoA = objValida.EvaluaSudoku(Individuo[indivA]);
        int valorIndividuoB = objValida.EvaluaSudoku(Individuo[indivB]);
        if (valorIndividuoA == 243 || valorIndividuoB == 243) break;

        //El mejor conjunto reemplaza al peor y la copia se muta
        if (valorIndividuoA > valorIndividuoB) {
            CopiaConjunto(indivA, indivB);
            Muta(indivB);
        }
        else {
            CopiaConjunto(indivB, indivA);
            Muta(indivA);
        }
    }

    //Después del ciclo, busca el mejor conjunto de individuos
    int ConjuntoMejor = 0;
    int MayorPuntaje = -1;
    for (int cont = 0; cont < Individuo.Count; cont++) {
        int valorConjunto = objValida.EvaluaSudoku(Individuo[cont]);
        if (valorConjunto > MayorPuntaje) {
            ConjuntoMejor = cont;
            MayorPuntaje = valorConjunto;
        }
    }
    return Individuo[ConjuntoMejor];
}

private void CopiaConjunto(int origen, int destino) {
    for (int cont = 0; cont < 81; cont++)
        Individuo[destino][cont] = Individuo[origen][cont];
}

private void Muta(int indiv) {
    int pos = azar.Next(81);
    int nuevo;
    do {
        nuevo = azar.Next(1, 10);
    } while (Individuo[indiv][pos] == nuevo);
    Individuo[indiv][pos] = nuevo;
}

public void ImprimeSudoku(int[] individuo) {
    //Pone el individuo (que es un arreglo unidimensional) dentro de un tablero de sudoku
    for (int cont = 0, columna = 0; cont < individuo.Length; cont++) {
        Console.Write(individuo[cont].ToString() + ",");
        columna++;
        if (columna == 9) {
            columna = 0;
            Console.WriteLine(" ");
        }
    }
}
}
}
}
}

```

```

//Algoritmo genético para generar Sudokus completos. Operador mutación

namespace Genetico11 {

    class ValidaSudoku {
        private int[,] tablero;

        //Constructor donde se crea el tablero de sudoku para hacer validaciones
        public ValidaSudoku() {
            tablero = new int[9, 9];
        }

        //Evalúa si el individuo es viable como sudoku
        public int EvaluaSudoku(int[] individuo) {
            int puntaje = 0;

            //Pone el individuo (que es un arreglo unidimensional) dentro de un tablero de sudoku
            for (int cont = 0, fila = 0, columna = 0; cont < individuo.Length; cont++) {
                tablero[fila, columna++] = individuo[cont];
                if (columna == 9) {
                    columna = 0;
                    fila++;
                }
            }

            //Evalua las filas
            for (int fila = 0; fila < 9; fila++)
                for (int num = 1; num <= 9; num++)
                    puntaje += NumeroFila(num, fila);
            //Console.WriteLine("Filas =" + puntaje.ToString());

            //Evalua las columnas
            for (int columna = 0; columna < 9; columna++)
                for (int num = 1; num <= 9; num++)
                    puntaje += NumeroColumna(num, columna);
            //Console.WriteLine("Columnas =" + puntaje.ToString());

            //Evalua los cuadros internos
            for (int fila = 0; fila <= 6; fila += 3)
                for (int columna = 0; columna <= 6; columna += 3)
                    puntaje += Cuadrointerno(fila, columna);
            //Console.WriteLine("Interno =" + puntaje.ToString());

            return puntaje;
        }

        //Retorna 1 si el número existe y no está repetido. 0 en caso contrario.
        private int NumeroFila(int num, int fila) {
            int cuenta = 0;
            for (int columna = 0; columna < 9; columna++)
                if (tablero[fila, columna] == num)
                    cuenta++;

            if (cuenta == 1) return 1;
            return 0;
        }

        //Retorna 1 si el número existe y no está repetido. 0 en caso contrario.
        private int NumeroColumna(int num, int columna) {
            int cuenta = 0;
            for (int fila = 0; fila < 9; fila++)
                if (tablero[fila, columna] == num)
                    cuenta++;

            if (cuenta == 1) return 1;
            return 0;
        }

        //Evalúa puntaje de cada cuadro interno
        private int Cuadrointerno(int fila, int columna) {
            int puntaje = 0;
            for (int num = 1; num <= 9; num++) {
                int cuenta = 0;
                if (tablero[fila, columna] == num) cuenta++;
                if (tablero[fila, columna + 1] == num) cuenta++;
                if (tablero[fila, columna + 2] == num) cuenta++;
                if (tablero[fila + 1, columna] == num) cuenta++;
                if (tablero[fila + 1, columna + 1] == num) cuenta++;
                if (tablero[fila + 1, columna + 2] == num) cuenta++;
                if (tablero[fila + 2, columna] == num) cuenta++;
                if (tablero[fila + 2, columna + 1] == num) cuenta++;
                if (tablero[fila + 2, columna + 2] == num) cuenta++;
                if (cuenta == 1) puntaje++;
            }
            return puntaje;
        }
    }
}

```

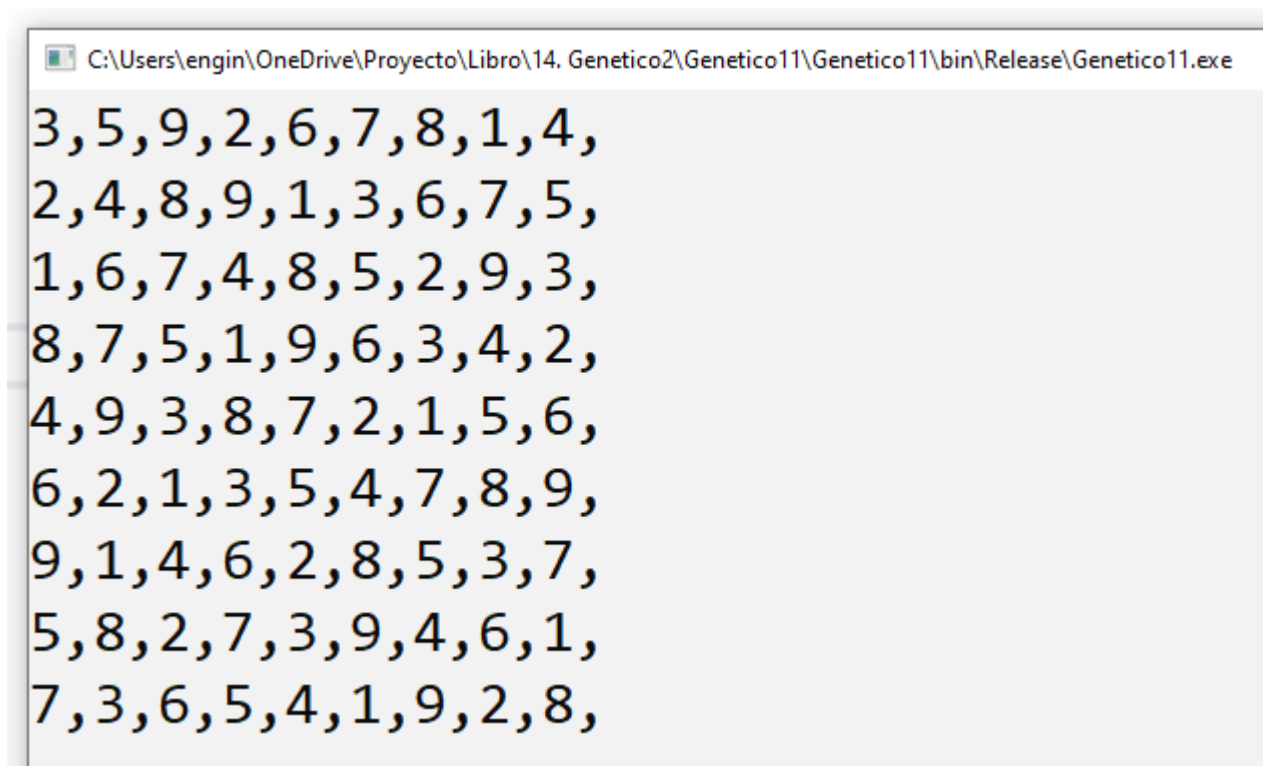



Ilustración 32: Tablero de Sudoku generado

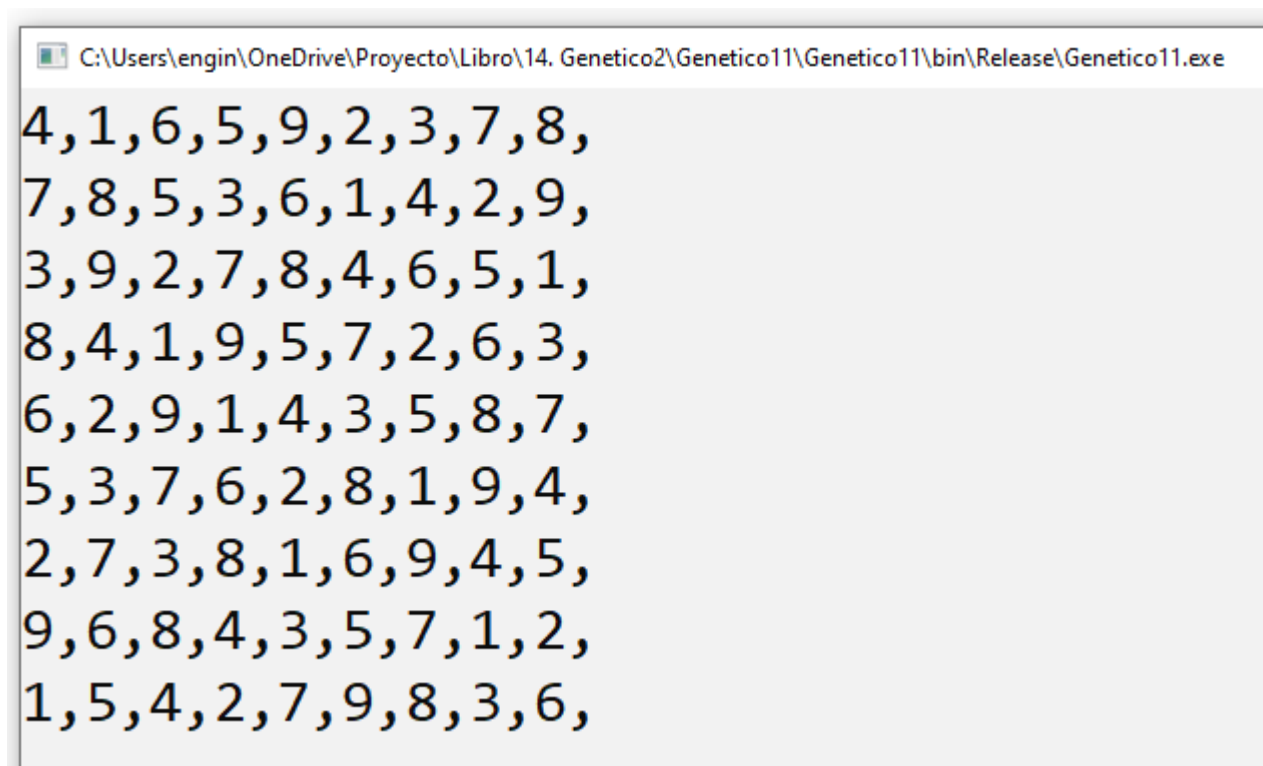


Ilustración 33: Tablero de Sudoku generado

Máximos y mínimos locales

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

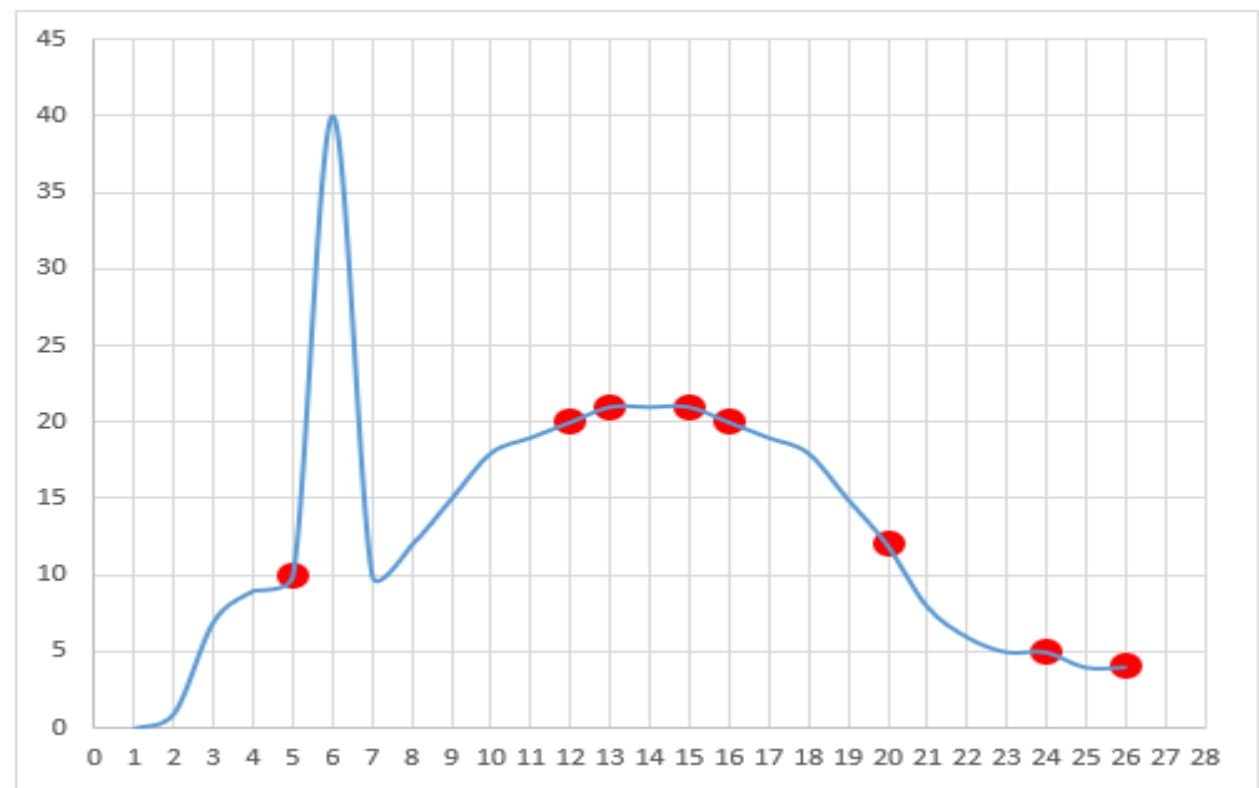


Ilustración 34: Gráfico matemático

Visualmente, el máximo “Y” se encuentra entre los valores de “X”: 5 y 7. Hay muchos individuos entre 11 y 17 que al competir contra el único individuo en la posición 5, ganarían y se reproducirían eliminando de la población al individuo que en verdad estaba cerca de la solución global. Lo más probable es que el algoritmo genético de más puntaje a los individuos entre 11 y 17, y se deduzca erróneamente que el máximo está entre 11 y 17. Eso se le conoce como máximo local y una vez que el algoritmo genético privilegie a los individuos de ese máximo local, se borrará cualquier esperanza de que se llegue al máximo global real.

¿Cómo lo solucionamos? 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 genético.
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. [4]

Variando el algoritmo genético

Algoritmo usado en este libro

Los algoritmos genéticos 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 lo reemplaza 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 genético 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 cope 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 genético 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, en el ejemplo de "Trazando el camino" se puede implementar que varíe dinámicamente el tamaño del genoma. Otros problemas como la regresión simbólica [5] que busca encontrar la expresión algebraica que explique el comportamiento de unos datos, el genoma que representa la expresión algebraica puede cambiar de tamaño dinámicamente.

Paralelizar el algoritmo

Los algoritmos genéticos son candidatos excelentes para ser paralelizados [6] [7] 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 genético en cada una.

Conclusiones

Los algoritmos genéticos es una técnica muy prometedora para encontrar respuestas muy aproximadas a la ideal en problemas complejos. Se inspiran en los procesos evolutivos que hay en la naturaleza, por lo tanto, se debe estar atento a las investigaciones sobre la evolución de las especies [8] [9] [10], porque servirán para mejorar o encontrar nuevos caminos algorítmicos que mejoren el desempeño y la obtención de mejores resultados.

Los algoritmos genéticos pueden ser usados en diversas áreas como la Seguridad Informática [11], el Arte [12], el Diseño Industrial [13], Acústica [14], Ingeniería Aeroespacial [15], Astronomía [16], Ingeniería Eléctrica [17], Mercados Financieros [18], Juegos [19], Geofísica [20], Ingeniería de Materiales [21] [22] [23], Matemáticas [24] [25], Militar [26] [27], Reconocimiento de patrones [28] [29], Diseño de rutas y horarios [30] o Diseño de máquinas [31] [32].

Los resultados obtenidos son muy buenos para la industria, luego es un campo de investigación activo donde los problemas deben plantearse como ambientes, las diversas soluciones a esos problemas se representan como individuos que pueblan esos ambientes y para encontrar nuevas y mejores soluciones (individuos), es concluir las reglas de selección, cruce y mutación.

Un campo de investigación muy llamativo es la Vida Artificial [33] y los algoritmos genéticos es una pieza clave [34]. La creación de modelos computacionales donde individuos se comporten como seres vivos incluyendo su evolución puede abrirnos a un mundo lleno de posibilidades para tener mejores aplicaciones (software) con menor tasa de errores, más eficientes, con mejor uso de recursos y un desarrollo mucho más rápido.

Bibliografía

- [1] Wikipedia, «GNU Lesser General Public License,» 2017. [En línea]. Available: https://es.wikipedia.org/wiki/GNU_Lesser_General_Public_License. [Último acceso: mayo 2020].
- [2] ©2020 Wolfram Alpha LLC, «WolframAlpha computational knowledge engine,» 28 abril 2020. [En línea]. Available: <http://www.wolframalpha.com/>. [Último acceso: mayo 2020].
- [3] Universidad de Salamanca. Dpto. Bioquímica y Biología Molecular., «(c) Genotipo y Fenotipo,» 2003. [En línea]. Available: <http://campus.usal.es/~dbbm/biotec/genfen.htm>. [Último acceso: mayo 2020].
- [4] Understanding Evolution, «Sexualidad y recombinación genética,» 06 Noviembre 2017. [En línea]. Available: https://evolution.berkeley.edu/evolibrary/article/_0_0/evo_22_sp. [Último acceso: mayo 2020].
- [5] R. A. Moreno Parra, «Programación genética: La regresión simbólica,» *Entramado*, vol. 3, núm. 1, pp. 76-85, 2007.
- [6] E. D. Cortes, «El problema de paralelizar,» 09 2009. [En línea]. Available: <http://www.lnds.net/blog/2009/09/el-problema-de-paralelizar.html>. [Último acceso: mayo 2020].
- [7] G. E. Blelloch y B. M. Maggs, «School of Computer Science, Carnegie Mellon University,» 2017. [En línea]. Available: <https://www.cs.cmu.edu/~guyb/papers/BM04.pdf>. [Último acceso: mayo 2020].
- [8] M. J. Landis y J. G. Schraiberz, «Punctuated evolution shaped modern vertebrate diversity,» 16 June 2017. [En línea]. Available: <https://www.biorxiv.org/content/biorxiv/early/2017/06/18/151175.full.pdf>. [Último acceso: mayo 2020].
- [9] R. Gras, «UWindsor researchers successfully simulate evolution of species,» noviembre 2017. [En línea]. Available: <http://www.uwindsor.ca/dailynews/2015-09-28/uwindsor-researchers-successfully-simulate-evolution-species>. [Último acceso: mayo 2020].
- [10] Futurism, «Breakthrough: Scientists See the Evolution of a New Species Occur in a Flask,» 2017. [En línea]. Available: <https://futurism.com/breakthrough-scientists-see-the-evolution-of-a-new-species-occur-in-a-flask/>. [Último acceso: mayo 2020].
- [11] A. Lazalde, «Inteligencia artificial: algoritmos genéticos para seguridad informática,» 25 Febrero 2012. [En línea]. Available: <https://hipertextual.com/2012/02/inteligencia-artificial-algoritmos-geneticos-para-seguridad-informatica>. [Último acceso: mayo 2020].
- [12] HiperTextual, «Mona Lisa, algoritmos genéticos y HTML5: la evolución de una sonrisa,» 09 Noviembre 2010. [En línea]. Available: <https://hipertextual.com/2010/11/mona-lisa-algoritmos-geneticos-y-html5>. [Último acceso: mayo 2020].
- [13] i.materialise, «Genetic Robots made with 3D printing without a human designer,» 30 November 2010. [En línea]. Available: <https://i.materialise.com/blog/genetic-robots-made-with-3d-printing-without-a-human-designer/>. [Último acceso: mayo 2020].
- [14] S. K. O. A. T. H. S. Y. A. y. H. K. Sato, «Applying genetic algorithms to the optimum design of a concert hall.,» *Journal of Sound and Vibration*. vol.258, no.3, pp. 517-526, 2002.
- [15] S. D. S. Y. T. y. N. H. Obayashi, «Multiobjective evolutionary computation for supersonic wing-shape optimization.,» *IEEE Transactions on Evolutionary Computation*, vol.4, no.2, pp. 182-187, 2000.
- [16] P. Charbonneau, «Genetic algorithms in astronomy and astrophysics.,» *The Astrophysical Journal Supplement Series*, vol.101, pp. 309-334, 1995.
- [17] E. y. D. L. Altshuler, «Design of a wire antenna using a genetic algorithm.,» *Journal of Electronic Defense*, vol.20, no.7, pp. 50-52, 1997.
- [18] S. y. G. M. Mahfoud, «Financial forecasting using genetic algorithms.,» *Applied Artificial Intelligence*, vol.10, no.6, pp. 543-565, 1996.
- [19] K. y. D. F. Chellapilla, «Evolving an expert checkers playing program without using human expertise,» *IEEE Transactions on Evolutionary Computation*, vol.5, no.4, pp. 422-428, 2001.
- [20] M. y. K. G. Sambridge, «Earthquake hypocenter location using genetic algorithms.,» *Bulletin of the Seismological Society of America*, vol.83, no.5, pp. 1.467-1.491, 1993.
- [21] R. M. C. y. D. G. Giro, «Designing conducting polymers using genetic algorithms.,» *Chemical Physics Letters*, vol.366, no.1-2, pp. 170-175, 2002.
- [22] D. U. H. y. T. B. Weismann, «Robust design of multilayer optical coatings by means of evolutionary algorithms.,» *IEEE Transactions on Evolutionary Computation*, vol.2, no.4, pp. 162-167, 1998.
- [23] F. A. O. E. M. O. H. y. W. B. Robin, «Simulation and evolutionary optimization of electron-beam lithography with genetic and simplex-downhill algorithms.,» *IEEE Transactions on Evolutionary Computation*, vol.7, no.1, pp. 69-82, 2003.
- [24] J. F. B. D. A. y. M. K. Koza, «Genetic Programming III: Darwinian Invention and Problem Solving,» *Morgan Kaufmann Publishers*, 1999.
- [25] M. Mitchell, «An Introduction to Genetic Algorithms,» *MIT Press*, 1996.

- [26] R. Kewley y M. Embrechts, «Computational military tactical planning system,» *IEEE Transactions on Systems, Man and Cybernetics, Part C - Applications and Reviews*, vol.32, no.2, pp. 161-171, 2002.
- [27] G. Naik, «Back to Darwin: In sunlight and cells, science seeks answers to high-tech puzzles.,» *The Wall Street Journal*, p. A1, 1996.
- [28] W.-H. Au, K. Chan y X. Yao, «A novel evolutionary data mining algorithm with applications to churn prediction,» *IEEE Transactions on Evolutionary Computation*, vol.7, no.6, pp. 532-545, 2003.
- [29] M. Rizki, M. Zmuda y L. Tamburino, «Evolving pattern recognition systems.,» *IEEE Transactions on Evolutionary Computation*, vol.6, no.6, pp. 594-609, 2002.
- [30] E. Burke y J. Newall., «A multistage evolutionary algorithm for the timetable problem,» *IEEE Transactions on Evolutionary Computation*, vol.3, no.1, pp. 63-74, 1999.
- [31] E. Benini y A. Toffolo, «Optimal design of horizontal-axis wind turbines using blade-element theory and evolutionary computation.,» *Journal of Solar Energy Engineering*, vol.124, no.4, pp. 357-363, 2002.
- [32] Y. Lee y S. H. Zak, «Designing a genetic neural fuzzy antilock-brake-system controller.,» *IEEE Transactions on Evolutionary Computation*, vol.6, no.2, pp. 198-211, 2002.
- [33] Wikipedia, «Vida artificial,» [En línea]. Available: https://es.wikipedia.org/wiki/Vida_artificial. [Último acceso: mayo 2020].
- [34] M. Mitchell y F. Stephanie, «Genetic Algorithms and Artificial Life,» 1993. [En línea]. Available: <https://www.santafe.edu/research/results/working-papers/genetic-algorithms-and-artificial-life>. [Último acceso: mayo 2020].