Universitat de les Illes Balears

escola politècnica superior

Treball Final de Grau

INGENIERÍA INFORMÁTICA, ESPECIALIDAD COMPUTACIÓN

Algoritmos genéticos

OSCAR ALONSO

Tutor
Margaret Miró

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, 14 de julio de 2017

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ACRÓNIMOS

AI Inteligencia Artificial

GA Algoritmos Genéticos

GUI Grafical User Interface

TFG Trabajo de Final de Grado

MSC Cifrado de Substitución Monoalfabetico

WPF Windows Presentation Foundation

XML eXtensible Markup Language

RESUMEN

Los algoritmos genéticos son unos algoritmos de optimización basados en la selección natural, donde las especies evolucionan mediante la reproducción a lo largo de las generaciones. Es un método de resolución muy flexible, que se puede usar en diferentes campos. Son muy potentes para solucionar problemas que para los cuales no existe una técnica de resolución específica, pero si para la resolución de un problema en concreto existen técnicas específicas, estos últimos suelen quedar por delante en eficiencia.

Este trabajo de fin de grado busca que el lector entienda cómo funcionan los algoritmos genéticos. Primero se presentarán los elementos básicos: los cromosomas, la reproducción, el "fitness", la mutación, el elitismo, la selección de progenitores... Sin embargo, se pondrá especial interés en la implementación de los mismos, con dos ejemplos totalmente distintos, que se desarrollarán y explicarán paso a paso.

Se ha decido llevar a cabo un primer problema en el campo registro de imágenes, cuya finalidad es la creación de una imagen panorámica, y un segundo problema en el ámbito de la criptología, más concretamente, el desencriptado de un texto, siendo este último más complejo de entender. Se ha optado por implementar estos dos problemas tan diferentes con la intención de demostrar su gran flexibilidad y potencia. A pesar de su gran capacidad de resolución, se probará que son más sencillos de programar y de entender de lo que parecen, de hecho, se puede observar que solo una clase se cambia para la resolución de ambos problemas, mientras que tanto las clases restantes como el esquema y la idea del funcionamiento del algoritmo son idénticos en los dos ejercicios. Sin embargo, si que se necesita un buen conocimiento del problema y del algoritmo en sí, ya que cualquier mínimo cambio puede afectar, tanto para bien como para mal, el desempeño del mismo.

Por otra parte, se presentarán las principales dificultades y limitaciones de estos algoritmos, así como sus ventajas, para que el lector tenga una idea de cuando aplicar aplicar este método es buena idea y cuando no, mostrando los resultados obtenidos de los ejercicios resueltos.

CAPÍTULO 1

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

Desde el principio de los tiempos, los humanos hemos intentado conocer, entender y controlar el mundo en el que vivimos. Durante todos estos años, hemos ido acumulando conocimiento útil que nos ha ayudado a sobrevivir como el control del fuego, la agricultura o la medicina.

Sin ser una excepción, entender el comportamiento y la mente humana ha sido un objetivo, desde el siglo VI a.c., de filósofos y psicólogos. El objetivo de la Inteligencia Artificial (AI) no es solo entender la mente humana, sino llegar a construir entidades inteligentes, llamadas agentes, que puedan actuar y pensar como un ser humano.

1.1 Inteligencia artificial en la resolución de problemas

1.1.1 Agentes

Se denomina agentes a cualquier cosa capaz de percibir su entorno, a través de unos sensores, y es capaz de reaccionar mediante los actuadores u operadores. Tal vez el ejemplo más claro de agente es un robot, que percibe el entorno mediante cámaras u otros sensores, y es capaz de desplazarse e incluso mover diversos elementos. Sin embargo, también se denomina agente a cualquier algoritmo o software, que es capaz de reaccionar a diversos elementos de entradas, como por ejemplo el teclado o el ratón, y actúa sobre el entorno mediante sus dispositivos de salida, como la pantalla, enviando datos por la red, etc.

La representación de toda la información de la que dispone un agente en un momento dado se denomina estado. A partir de esta información el agente reaccionará, cambiando su entorno. Si estos agentes intentan hacer lo mejor, es decir, obtener la mejor de las salidas, se les llaman agentes racionales.

1.1.2 Resolución de problemas y búsqueda

Uno de los puntos más importantes en los cuales se aplica la inteligencia artificial es la resolución de problemas. Ésto se presenta como una búsqueda en un conjunto de estados y la AI propone varias técnicas para resolverlos. Para que problema se pueda resolver se debe formular definiendo, al menos, cuatro partes [1]:

- *Estado inicial:* Datos que el agente conoce en el momento en el cual el agente empieza.
- *Función objetivo:* Función que indica si el estado en el que se encuentra es el objetivo final.
- *Acciones:* Listado de operaciones que permiten al agente pasar de un estado a otro. En la mayoría de problemas de este tipo, pasar de un estado a otro suele tener un coste asociado.
- *Espacio de búsqueda:* Todos los estados que se pueden alcanzar mediante las acciones.

Se llama búsqueda al análisis de los estados con intención de llegar al objetivo. El agente va probando todos los estados disponibles, creando un camino hasta que llega a la solución. Normalmente buscan aquel camino que tiene un coste más bajo asociado, para ello, se debe comprobar todos y cada uno de los estados existentes y disponer de una función o elemento que devuelva este valor asociado. Cuando se intenta solucionar un problema, normalmente el espacio de búsqueda es tan amplio que resulta imposible analizar todos y cada uno de los estados. Por eso mismo, se han desarrollado muchas técnicas que permiten reducir este numero de estados, ignorando aquellos que ya supone que no le van a servir como solución. Esta función que guía al algoritmo de búsqueda se le llama huerística. Muchas veces, la aplicación de esta técnica hace que la solución no sea la óptima, aunque si muy cercana a ella.

1.1.3 Optimización

La optimización es otro campo de la AI muy similar a la búsqueda, de hecho también se le llama búsqueda local. La principal diferencia entre estos es que en la optimización no importa el recorrido o camino hasta la solución, si no el estado final en si.

Consiste en la selección del mejor elemento, con respecto a algún criterio establecido, de un conjunto de elementos disponibles. Saber cual es el mejor requiere dé una función que nos dé un valor de puntuación, algo que no es fácil de establecer.

Por otra parte, buscar en todo el espacio de búsqueda es imposible, así que volvemos a requerir de algo que nos permita reducir los estados a comprobar, para redecir el tiempo necesario de ejecución. Por si fuese poco, un gran número de algoritmos propuestos para resolver problemas no son capaces de hacer una distinción entre soluciones óptimas locales y soluciones óptimas totales, y tratan a las primeras como soluciones del problema original. Como veremos más adelante, lo GA intentan evitar que ambas cosas sucedan.

CAPÍTULO 2

ALGORITMOS GENÉTICOS

2.1 Introducción

Los GA es un método de optimización iterativo basado en la teoría de Darwin de la evolución. Las especies evolucionan con cada generación, donde, debido a la selección natural, solo aquellos organismos más aptos para la supervivencia son capaces de transmitir sus genes a sus descendientes mediante la reproducción.

De forma similar, el algoritmo se inicia con un conjunto de estados, que se llaman individuos. Este conjunto va evolucionando a lo largo de numerosas iteraciones, aplicando diversos operadores, con la intención de alcanzar nuevos estados, mejores que los existentes.

Un operador es la reproducción, que se lleva a cabo escogiendo dos individuos que harán la función de padres, de los cuales saldrá uno o más hijos, es decir, uno o mas estados. Además, de la misma forma que pasa con la presión selectiva de la teoría de selección natural de Darwin, también se debe dar prioridad a seleccionar como padres a los mejores individuos que otorgarán sus genes, en este caso sus valores, a los hijos, favoreciendo la correcta evolución de los individuos, convergiendo a una solución óptima.

Otro operador que permite explorar nuevos estados es la mutación, también presente en la teoría de Darwin. Éste, realiza cambios al azar sobre los valores de los individuos.

Para finalizar, se establece un criterio de parada, que indicará cuando un estado es tomado por válido. Como no se asegura que este criterio sea alcanzado, también se puede establecer un máximo de iteraciones a realizar antes de que se acabe el algoritmo.

La figura 2.1 muestra el flujo básico que sigue un algoritmo genético. Todas y cada una de las operaciones y elementos se explicarán en el siguiente sección, y será el flujo que seguirán los ejercicios propuestos.

Aunque este tipo de algoritmos no siempre encuentra la solución óptima, si que se aproxima bastante, evitando caer en óptimos locales. Ademas, a pesar de tener una

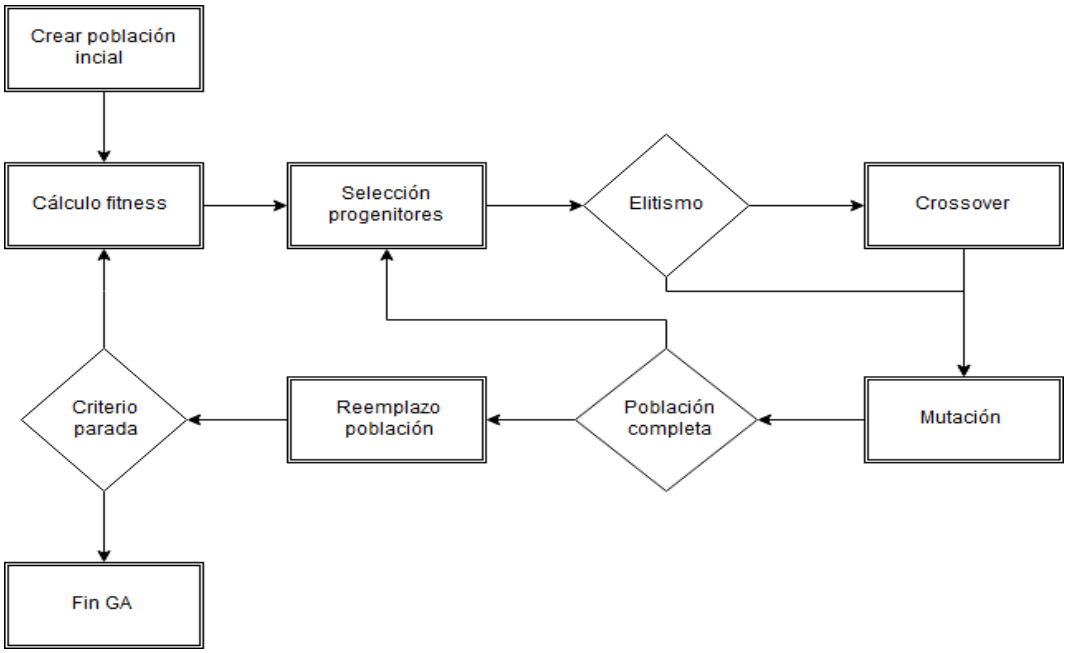

Figura 2.1: Flujo básico algoritmo genético

gran capacidad de resolución y robustez, se puede aplicar en la mayoría de problemas y son bastante fáciles de entender y de programar.

2.1.1 Elementos básicos de un algoritmo genético

Cromosomas

También llamado criatura, genotipo o individuo. Cada una de los estados dentro del espacio de búsqueda. Normalmente está representado como un conjunto de 0's y 1's que conforman un string binario o "array", sin embargo, hay otras formas de representación de los datos, como pueden ser los valores alfanuméricos.

Gen

Cada uno de los elementos que conforman el "array" que representa el cromosoma, es decir, cada uno de los 0's y 1's. En el caso de que los individuos estén representados con valores alfanuméricos, cada una de las variables será un gen.

Población

Conjunto de los cromosomas que se están estudiando actualmente. El tamaño de la población se especificará al iniciar el algoritmo y no se modificará. Para un mismo GA, cambiar el tamaño de la población puede afectar en gran medida los resultados del mismo.

Generación

Un algoritmo genético es un método iterativo. En cada iteración se crean y destruyen cromosomas mediante una serie de operadores, para crear una nueva población. Estas iteraciones se llaman generaciones.

"Fitness"

En castellano similitud, aunque es más usado el termino inglés. Es una función o valor asociada a cada cromosoma que se usa para medir la puntuación del mismo. Cuanto más alta sea este valor (o menor, según como se definan las operaciones), mejor será la solución que representa.

2.1.2 Operadores básicos de un algoritmo genético

Selección de progenitores

Operador que selecciona los individuos de la población para la reproducción (más adelante explicada). La selección de los progenitores tiene muchas variantes, pero las más eficientes y ampliamente usadas se basan en el valor "fitness" calculado que tienen los individuos, dando más prioridad a aquellos que son "mejores" [2].

Métodos de selección

- *Al azar*: Se seleccionan los cromosomas totalmente al azar. Esta forma es de las más simples y dado que no tiene en cuenta el fitness es normalmente descartada.
- *Truncado*: Solo se toma un porcentaje de los mejores a la hora de seleccionar los individuos para reproducirse. Por ejemplo, teniendo una población de 100 y un método de selección por truncado de un 20 % se ordena la población según el "fitness" y se guardan los primeros 20 individuos. Los 80 restantes son eliminados. La selección se haría de forma al azar sobre estos 20 individuos. Aunque es mejor que el simple ya que se tiene en cuenta el "fitness", se pierde demasiada variedad debido al descarte, así que tampoco es muy usada.
- *Selección de la ruleta*: Con este método, los individuos con mayor "fitness" tienen también más posibilidad de ser seleccionados. La mejor forma de entenderlo es pensando en un ruleta, donde cada individuo ocupa una fracción directamente proporcional a su "fitness". Esta porción se suele calcular dividiendo el fitness del individuo entre el "fitness" total ($\frac{fitness_i}{\text{fitness total}}$). Se hace girar la rueda y, cuando ésta pare, se selecciona aquel en cuya porción se encuentre el indicador. De esta forma, los mejores individuos tienen una fracción de la ruleta mayor y por lo tanto mayor posibilidad de ser elegidos, pero sin provocar que aquellos no tan buenos no puedan reproducirse,
- *Ranking lineal*: Este método es muy similar al anterior, funcionando como si de una ruleta se tratará. Sin embargo, la porción otorgada a cada individuo no se calcula directamente según el "fitness" sino que depende de la posición que ocupa en un ranking. Los individuos se ordenan de menos a más según su "fitness" y su posición en el mismo indicará que porcentaje de la ruleta tendrá ($\frac{posición_i}{\sum_{i=1}^{n} posición_i}$).

De esta forma las posibilidades de que cada individuo sea seleccionado están normalizadas. En una población con 5 individuos, el que tiene mayor "fitness" tendrá un probabilidad de $\frac{5}{5+4+3+2+1} = 0{,}33$ de ser seleccionado, mientras que el peor $\frac{1}{5+4+3+2+1} = 0{,}067$

- *Selección por torneo*: Aunque bastante simple de implementar, tiene muy buenos resultados. Se seleccionan X individuos completamente al azar, sin repeticiones, y de estos se selecciona el mejor. Dependiendo de la cantidad de individuos que forman cada "fase" del torneo afectamos la probabilidad de que los cromosomas peores sean o no seleccionados. Si este valor es 1, todos tendrán la misma probabilidad de ser escogidos, si el valor es muy grande, los peores nunca (o casi nunca) serán seleccionados.

"Crossover"

También llamado recombinación o reproducción, este operado se encarga de, a partir de dos (o más) progenitores, crear un nuevo individuo, que será introducido en la población que formará la siguiente generación. Hay que tener en especial cuidado con los métodos de selección y reproducción, ya se podría perder diversidad en los genes de los individuos. La reproducción le da al GA la posibilidad de explorar todo el espacio de búsqueda.

Métodos de "crossover"

- *Recombinación en un punto*: Se escoge un punto al azar en los dos progenitores. Todos los genes del nuevo individuo hasta el punto provendrá de uno de los dos cromosomas originales, el resto proviene del otro. Si se forma un segundo hijo, este es el inverso, es decir, contiene los genes que no se han seleccionado.

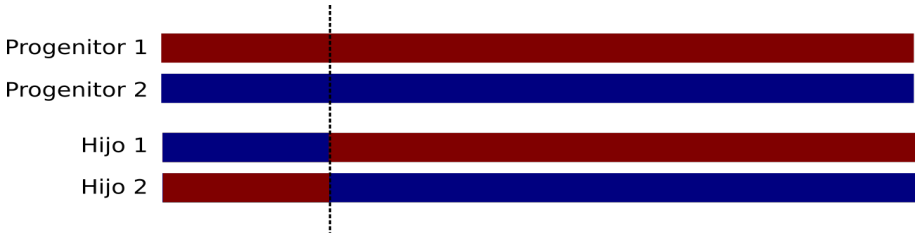

Figura 2.2: Recombinación en un punto

- *Recombinación en dos puntos*: Se escogen dos puntos al azar. Todos los genes del cromosoma nuevo hasta el primer punto provendrán de uno de los dos progenitores, del primer punto al segundo, del otro individuo progenitor, el resto del primero otro vez. Igual que en el anterior, si se forma un segundo hijo, este es el inverso, es decir, contiene los genes que no se han seleccionado.

- *Recombinación uniforme*: Cada gen de los nuevos individuos se selecciona al azar de entre los dos progenitores con una probabilidad de 0,5. Si se crean dos individuos, el gen no seleccionado se copia en el segundo hijo.

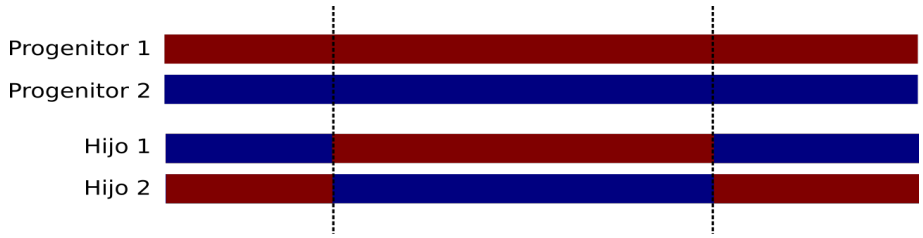

Figura 2.3: Recombinación en dos punto

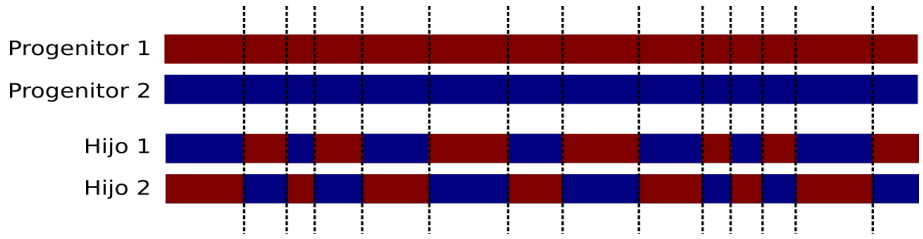

Figura 2.4: Recombinación uniforme

- *Recombinación uniforme media*: Es igual que el uniforme, pero asegura que los hijos tiene exactamente un 50 % de los genes no comunes de cada progenitor.
- *Recombinación de tres progenitores*: En este caso solo se puede generar un hijo, que estará formado por aquellos genes que esta presente en al menos dos de los progenitores.
- *Recombinación no binaria*: Aunque los casos comentados están pensados para cromosomas representados por datos binarios, es decir, un "array" de 0's y 1's, se pueden aplicar para individuos no representados de tal forma, donde cada gen es un variable. Si por ejemplo los genes de un individuo son 5 variables, se podría crear un nuevo individuo aplicando el método uniforme, escogiendo al azar uno de los valores para esa variable.

No "crossover"

Aunque la idea principal del algoritmo es que la generación actual se reproduzca para producir una nueva población "evolucionada", puede interesar que algunos individuos no se recombinen y pasen a la siguiente generación con los mismos genes. Esto afectan a la velocidad con la que cambia las poblaciones. Si el ratio de individuos que no se reproducen es muy grande, se necesitarán muchas mas generaciones para completar el espacio de búsqueda. Por otra parte, si todos los individuos se reproducen, pueden perderse valores, produciéndose unos saltos muy grandes entre generaciones.

Elitismo

Se llama élite a un grupo de individuos que tiene una característica superior al resto y por ello, suele tener un trato especial. El elitismo es una versión de la no reproducción,

en la cual, los X mejores elementos de la población actual son pasados a la siguiente generación, aplicando, o no, la mutación. Estos, pueden ser luego seleccionados como progenitores para crear un nuevo individuo. Este método asegura que la siguiente generación no perderá los valores buenos que se han descubierto. Ayuda también a una buena convergencia de la población, siempre y cuando se haya estudiado y se controle, si no puede conllevar a una mala evolución [3].

Mutación

Este operador cambia 0's por 1's, y viceversa, en los genes de los individuos. Su finalidad es crear individuos que no se podrían crear mediante la reproducción debido a la falta de diversidad de los genes que conforman la población. A diferencia de la reproducción, no asegura que los individuos creados evolucionen hacia la solución óptima, solo pretende premitir que el espacio de búsqueda sea explorado por completo. No se deben mutar todos los genes, de hecho, la probabilidad de que se produzca una mutación es menor que el 0,01 % en la mayoría de problemas. Sin embargo, este ratio de mutación puede depender de otros elementos de la configuración del GA. Si por ejemplo, la población es muy pequeña, la probabilidad de mutación deberá ser más alta para compensar la falta de variedad. La mutación también proporciona la posibilidad de explorar todo el espacio de búsqueda, aunque con pasos más pequeños.

2.2 Ejemplo simple de algoritmo genético

2.2.1 Planteamiento

Una vez presentado que es un GA y sus componentes, planteamos un ejercicio muy simple con la idea de entender, de forma muy superficial, como funcionan.

En este problema, nos encontramos con diversos objetos que tienen un valor (€) y peso (kg) asociado. Disponemos de un coche (o cualquier otro vehículo) con el cual queremos transportar los objetos, en un solo viaje y maximizando el valor de las piezas transportadas. Sin embargo, aunque el espacio no supone una limitación en este caso, si lo es el peso que podemos transportar. Cada kilo que transportemos nos costará 1 €en gasolina. Debido a que nuestro coche es muy viejo, si superamos el peso máximo que soporta, el coste que nos cuesta transportar cada kilogramo extra será de 3 €.

Aunque este problema se podría resolver de muchas otras formas, lo he elegido debido a su simplicidad.

2.2.2 Creación de la población

Representación de los cromosomas

Quizás lo más difícil a la hora de solucionar un problema mediante un algoritmo genético es establecer una estructura de datos que se acomode a nuestro problema.

Debido que quiero mantener la simplicidad en este ejemplo, he optado por un conjunto de "arrays" que representan tanto a los individuos en sí como a la población total. Probablemente ésta no sea (ni de cerca) la mejor forma de tratar los datos, pero la introducción de clases en este punto podría complicar demasiado las cosas.

Se define por individuo a un caso en el cual X objetos son trasportados en el vehículo. Cada individuo pues, es un “array” de enteros de longitud “LEN” (que corresponde al número de objetos que hay en nuestro problema). Suponiendo que cada objeto esta etiquetado del 0 a “LEN”, si la correspondiente posición en el “array” a esta etiqueta es igual a 1, significará que esta siendo transportado, si es 0, no. Cada una de estas posiciones es un gen del individuo.

La población es simplemente un agregado de todos los individuos (es decir, un “array” bidimensional) que se están estudiando en esta generación.

El resto de variables son triviales y no merecen ser explicados.

```
1  //Numero de la población (DE MOMENTO LA MITAD DE LA POP TIENE QUE SER PAR)
2  public const int POP = 32;
3  //longitud del genotipo (Individuo)
4  public const int LEN = 15;
5
6  //Array que contiene el peso de los objetos
7  public static int[] weightArray = new int[LEN];
8  //Array que contiene el valor de los objetos
9  public static int[] valueArray = new int[LEN];
10
11 //Array bidimensional con los individuos
12 public static int[][] population = new int[POP][];
13 //Donde la nueva población es guardada mientras se construye
14 public static int[][] newPopulation = new int[POP][];
15 //Array que contiene el valor "fitness" de los individuos
16 public static int[] fitness = new int[POP];
17 //Array que indica que individuos son seleccionados para realizar el "crossover"
18 public static int[] selection = new int[POP];
19 //Valor del mejor individuo
20 public static int bestFitness = -2147483648;
21 //Mejor individuo encontrado
22 public static int[] bestIndividualFound = new int[LEN];
23 //Generacion actual
24 public static int generation = 0;
```

También definimos todas las constantes, que nos facilitan jugar y probar diferentes combinaciones para ver como se comporta el algoritmo. Estas no tiene nada que ver con la definición de los individuos o la población.

```
1  //Clase de C# para obtener nuemero pseudoaleatorios
2  public static Random rnd = new Random();
3
4
5  //Numero de "fitness" individuos seleccionados
6  public const int BESTS = 13;
7  //Generaciones a correr
8  public const int GENERATIONS = 15;
9  //Probabilidad de mutación (%)
10 public const int PROB = 1;
11 //Maximo del peso soportado
12 public const int MAXWEIGHT = 150;
```

Inicializar la población

De forma aleatoria se crea la población con la que empezaremos nuestro problema. Para ello se recorre cada uno de los genes de todos los individuos y se escribe un 1 o un 0 (el objeto es transportado o no, respectivamente) de forma aleatoria. Para favorecer la diversidad de población, la probabilidad de que un gen tenga un valor u otro debería ser la misma.

Si la probabilidad de escribir un 1 es muy baja puede suceder que no se cree ningún individuo que, por ejemplo, transporte el primer objeto. Si esto sucede, ninguno de los individuos de las siguientes generaciones podrán tampoco transportar este objeto a no ser que se produzca un cambio mediante la mutación. Por lo tanto, si le añadimos que la probabilidad de mutación sea 0 o casi nula, nunca, en todas las generaciones, existiría un individuo que transportase ese objeto, y el espacio de búsqueda no estaría completo. Puede ser que la solución óptima sea incluyendo ese objeto, pero nunca será estudiado este caso. Esto se puede ver afectados por muchas otras variables. Si la población es muy grande, habrá más diversidad y más probabilidad de que un gen perdido reaparezca. Lo mismo si el número de generaciones que se producen es mayor.

En este caso, se ha decidido que la probabilidad de escribir un 0 sea mucho mayor que la probabilidad de escribir un 1, sin embargo, gracias al tamaño de la población, numero de generaciones que se llevará a cabo y la probabilidad de mutación aseguran que ninguno de los genes se pierda.

Aunque no lo parezca, la buena inicialización y representación de los datos afecta al desarrollo del problema, tanto al resultado final como al tiempo necesario para alcanza la solución óptima o cercana.

```
1  public static void initPop() {
2    //Por cada individuo
3    for (int i = 0; i < POP; i++) {
4      //Instanciamos los array para poder llenarlos
5      population[i] = new int[LEN];
6      newPopulation[i] = new int[LEN];
7      //Cada gen del individuo
8      for (int j = 0; j < LEN; j++) {
9        //Mas posibilidades de 0
10       if (rnd.Next(0, 9) < 5) {
11         population[i][j] = 0;
12       } else {
13         population[i][j] = 1;
14       }
15     }
16   }
17 }
```

2.2.3 Calculo del "fitness" de cada individuo

Calcula la similitud de cada cromosoma para, la hora de generar la siguiente generación, escoger como progenitores aquellos individuos que mejor solucionan nuestro problema. Calcular el valor "fitness" de nuestros individuos es tan fácil como mirar el valor y el peso que tienen los objetos que lo componen. Se suma el valor que proporcionan todos ellos y, por otra parte, el peso que tienen. Si el peso total de los objetos supera el peso máximo, hay que multiplicar por 3 el exceso para obtener el coste de transporte.

Finalmente, restamos este coste al valor de los objetos transportados y, el resultado es el valor “fitness” de nuestro individuo.

En esta misma función se aplica el elitismo, por motivos de eficiencia, para aprovechar la ordenación de los individuos. El método esta aplicado en la sección correspondiente 2.2.6.

```
1  public static void calcFitness()
2  /*
3  Miramos el "fitness" de cada individuo
4  La finalidad es conseguir un mayor valor, teniendo en cuenta que si se sobre
       pasa el peso maximo supone un coste extra
5  */
6  {
7    int value = 0;
8    int weight = 0;
9    for (int ind = 0; ind < POP; ind++) {
10     value = 0;
11     weight = 0;
12     //Obtenemos el valor "fitness"
13     for (int gen = 0; gen < LEN; gen++) {
14       if (population[ind][gen] == 1) {
15         value = value + valueArray[gen];
16         weight = weight + weightArray[gen];
17       }
18     }
19     //Comprobamos si el peso se pasa del mÃ¡ximo
20     fitness[ind] = value;
21     if (weight < MAXWEIGHT) {
22       fitness[ind] = fitness[ind] - weight;
23     } else {
24       weight = weight - MAXWEIGHT;
25       fitness[ind] = fitness[ind] - MAXWEIGHT;
26       fitness[ind] = fitness[ind] - weight * 3;
27     }
28     //Aplicacion de elitismo: si es el mejor individuo lo guardamos
29     if (bestFitness < fitness[ind]) {
30       bestFitness = fitness[ind];
31       for (int gen = 0; gen < LEN; gen++) {
32         bestIndividualFound[gen] = population[ind][gen];
33       }
34     }
35   }
36 }
```

2.2.4 Selección

Selecciona los cromosomas que funcionarán como progenitores a la hora de crear nuevos individuos. Como hemos comentado anteriormente, seleccionamos aquellos individuos que más nos conviene. Sin embargo, si escogemos los que tienen el valor de “fitness” más alto se podría perder en diversidad. Por eso mismo se escoge al azar un número de individuos entre aquellos que tienen el “fitness” más bajo.

Ordena en un nuevo “array” los mejores individuos, el número de estos viene indicado por la variable “BEST”. Después, escoge el resto, hasta completar el tamaño

de población, al azar (para evitar perder variedad). Para ello se ayuda de una función que determina cuando un individuo ya ha sido seleccionado.

```
1  public static void calcSelection()
2  /*
3  Escoge los individuos que participarán en el ''crossover''
4  */
5  {
6    for (int i = 0; i < BESTS; i++) {
7      int maxValue = -2147483648; // Int mas bajo repesentable
8      int maxIndex = 0;
9      for (int j = 0; j < POP; j++) {
10       if (fitness[j] > maxValue && !isInSelection(j)) {
11         maxValue = fitness[j];
12         maxIndex = j;
13       }
14     }
15     selection[i] = maxIndex;
16   }
17
18   for (int i = BESTS; i <= POP / 2; i++) {
19     int randomIndividual = rnd.Next(0, POP);
20     while (isInSelection(randomIndividual)) {
21       randomIndividual = rnd.Next(0, POP);
22     }
23     selection[i] = randomIndividual;
24   }
25 }
26
27 public static bool isInSelection(int index)
28 /*
29 Comprueba si el individuo ya esta entre los seleccionados
30 */
31 {
32   for (int i = 0; i < POP; i++) {
33     if (selection[i] == index) {
34       return true;
35     }
36   }
37   return false;
38 }
```

2.2.5 "Crossover"

Efectúa la reproducción entre dos cromosomas, que forman un nuevo individuo, compartiendo genes de ambos padres.

Seleccionamos uno a uno los individuos que conforman el "array selection", y le buscamos al azar una pareja, que no puede ser ella misma. Una vez tenemos los dos progenitores hacemos la reproducción para crear dos nuevos individuos.

La reproducción, dado que se quiere mantener simple todo el conjunto del problema, se basa también en el azar, utilizando el método de recombinación en un punto. Recordamos como funciona esta forma de "crossover". Un número aleatorio establece cuantos genes de un individuo conformarán el nuevo, los elementos restantes pertene-

cen a su pareja. De la misma forma, se crea un segundo individuo que tendrá los genes no escogidos, es decir, el inverso.

En este ejemplo no hay una probabilidad como tal de que no se produzca reproducción, sin embargo, si el numero de genes a copiar es igual a 0, los dos nuevos individuos serán exactamente igual que sus padres. La probabilidad de que no haya "crossover" es entonces de $\frac{1}{\text{Nº genes del individuo}}$.

```
1  public static void crossOver()
2  /*
3  Efectua la reproducción entre dos individuos (se puede no producir, crossLimit =
       0 )
4  */
5  {
6    int newInd = 0;
7    for (int mateIndex = 0; mateIndex < POP / 2; mateIndex++) {
8      int male = mateIndex;
9      int female = selectMate(male);
10     int crossLimit = rnd.Next(0, LEN);
11
12     for (int gen = 0; gen < LEN; gen++) {
13       if (gen < crossLimit) {
14         newPopulation[newInd][gen] = population[male][gen];
15         newPopulation[newInd + 1][gen] = population[female][gen];
16       } else {
17         newPopulation[newInd][gen] = population[female][gen];
18         newPopulation[newInd + 1][gen] = population[male][gen];
19       }
20     }
21     newInd = newInd + 2;
22   }
23 }
24
25 public static int selectMate(int male)
26 /*
27 Escoge un individuo que efectuará la "reproducción"
28 */
29 {
30   int mate = rnd.Next(0, POP / 2);
31   while (selection[mate] == male) {
32     mate = rnd.Next(0, POP / 2);
33   }
34   return selection[mate];
35 }
```

Este método lo único que hace es buscar entre todos los seleccionados una pareja para el individuo dado. Aunque ahora el método es bastante simple y solo comprueba que no sea el mismo, se pueden hacer mejores búsquedas y heurísticas que podremos ver en los siguientes ejemplos (y que ya se han explicado un poco por encima en el punto 2.1.2).

```
1  int selectMate(int male)
2  /*
3  Escoge un individuo que efectuará la "reproducción"
4  */
5  {
6    int mate = rand() % (POP / 2);
```

```
7   while (selection[mate] == male){
8     mate = rand() % (POP / 2);
9   }
10  return selection[mate];
11 }
```

2.2.6 Elitismo

En esta práctica aplicamos el método más sencillo de elitismo, guardar el mejor individuo encontrado. El código esta dentro de la función de cálculo de "fitness", que podemos ver en la sección 2.2.3. Por cada generación, aprovechando que calculamos su "fitness", miramos si este es el mejor cromosoma encontrado, si es así, lo guardamos en una variable global llamada "bestIndividualFound". Cuando el algoritmo acabe, el individuo que este en esta variable, será la solución al problema.

2.2.7 Mutación

Realizar el proceso de mutación es muy sencillo. Recorremos todos y cada uno de los genes de los individuos y cambiamos un 0 por un 1 o viceversa. El gen solo muta si un número calculado al azar es menor que la probabilidad establecida.

```
1  public static void mutation()
2  /*
3  Realiza la mutacion
4  */
5  {
6    for (int ind = 0; ind < POP; ind++) {
7      int prob;
8      for (int gen = 0; gen < LEN; gen++) {
9        prob = rnd.Next(0, 100);
10       if (prob < PROB) {
11         newPopulation[ind][gen] = Math.Abs(newPopulation[ind][gen] - 1);
12       }
13     }
14   }
15 }
```

2.2.8 Remplazo de la población

Una vez tenemos la nueva población, solo hay que establecerla como la población de estudio.

```
1  public static void replace()
2  /*
3  Replaza las poblaciones
4  */
5  {
6    for (int ind = 0; ind < POP; ind++) {
7      for (int gen = 0; gen < LEN; gen++) {
8        population[ind][gen] = newPopulation[ind][gen];
9        newPopulation[ind][gen] = 0;
10     }
11     fitness[ind] = 0;
```

```
12      selection[ind] = 0;
13    }
14  }
```

2.2.9 Criterio de parada

En este ejercicio tan simple, el único criterio de parada que hay es cuando se alcanzan las generaciones establecidas. Aunque esto puede provocar que el algoritmo se siga ejecutando cuando ya no hay una solución mejor, en algunos caso se puede no tener forma de detectar cuando una solución es suficientemente buena, o simplemente interesa más arriesgarse y dejar que siga buscando.

2.2.10 Programa principal

El programa principal solo hace las llamadas a las funciones en el orden adecuado. Después de inicializar la población y los valores de los objetos repite tantas veces como generaciones se desee la selección de individuos, crossover, la mutación y empieza de nuevo con la nueva población.

```
1  public static void Main() {
2    initProgram();
3    initPop();
4    do {
5      calcFitness();
6      calcSelection();
7      crossOver();
8      mutation();
9      replace();
10     generation++;
11   } while (generation < GENERATIONS);
12 }
```

2.3 Código de un algoritmo genético

Aunque escribir un GA no es muy complicado, existen librerías que facilitan la programación y desarrollo de aplicaciones que hacen uso de algoritmos genéticos. Algunas librerías son:

- *ECJ*: Para java
- *Jenetics*: Para java
- *pyevolve*: Para python
- *aforge*: Para C♯
- *GAlib*: Para C++
- *Evolving Objects*: Para C++
- *Open Beagle*: Para C++

La mayoría de estas librerías te obligan a sobrescribir la representación y los métodos de los cromosomas, mientras que todo lo demás te lo administra ella. Aunque todas dejan elegir tamaño de población, probabilidades de mutación y demás cosas básicas, algunas son más con figurables, permitiéndote especificar el uso de métodos más avanzados, como sobrescribir el método de selección de progenitores.

Sin embargo, la intención de este documento es enseñar como funcionan internamente estos algoritmos, así no se hará uso de ninguna de estas librerías, sino que escribirán todos los elementos y funciones con tal de explicar y entender los algoritmos genéticos.

2.4 Resumen

En este capítulo se ha hecho una introducción a los GA, y se han presentado sus operaciones fundamentales: la selección, reproducción y mutación, así como sus principales elementos, comparándolos con la teoría de evolución de Darwin, en la cual se basan este tipo de algoritmos.

También se ha podido ver el código de un ejemplo comentado, explicando cada uno de los detalles e implementaciones, con la intención de demostrar que escribir un algoritmo genético es de lo más sencillo, con tan solo unas 200 líneas de código nos ha permitido resolver el problema propuesto.

A pesar de su facilidad y su sencillez, los algoritmos genéticos han demostrado ser capaces de poder resolver un gran número de problemas. Algunos ejemplos son:

- *Registro de imagen*
- *Medicina*
- *Robótica*
- *Aeronáutica*
- *Reconocimiento facial*
- *Criptología*
- *Entrenamiento y diseño de redes neuronales*
- *Evolución de software*
- *Teoria de juegos*

Los algoritmos genéticos funcionan en todas estas aplicaciones por dos simples razones. Primero, el espacio de búsqueda es demasiado grande como para comprobar todas y cada una de las posibilidades. Segundo, verificar si una solución es correcta o no es relativamente sencilla [4].

En los siguientes capítulos se van presentar dos problemas, los cuales se resolverán mediante los algoritmos genéticos. El primero de ellos es el registro de imágenes, en el cual se deberá formar una panorámica a partir de dos fotografías. En segundo, trata sobre la criptología, exactamente, en la capacidad de desencriptar un texto.

CAPÍTULO 3

ALGORITMOS GENÉTICOS APLICADO AL REGISTRO DE IMÁGENES

3.1 Registro de imágenes

El registro de imágenes consiste en transformar dos fotografías en un mismo sistema de coordenadas. Esto supone realizar desplazamiento, rotaciones y escalados para lograr que ambas imágenes se superpongan de la mejor forma posible. Es un proceso computacionalmente alto, ya que hay demasiadas variables a tener en cuenta: las fotografías pueden haberse realizado desde distintos ángulos, con distinta iluminación, la zona de solapamiento puede ser un extremo o en el centro, etc, etc... Este proceso se lleva a cabo en ámbitos sobretodo en la medicina, donde ayuda al medico a aplicar un diagnóstico y tratamiento e incluso, asistir durante la cirugía. Otras aplicaciones es el reconocimiento facial, reconstruir áreas mediante imágenes de satélites (cartografía) o el reconocimiento espacial en la robótica.

Otra aplicación, la cual se va a explicar e implementar a lo largo de este capítulo, es la creación de una imagen panorámica. A partir de dos fotografías de la misma escena, se tendrá que calcular la posición en la cual las imágenes forman una sola, con el menor número de interferencias o distorsión.

3.1.1 Operaciones básicas

La informática gráfica, la robótica, visión por computador y demás áreas que tratan el tema de figuras e imágenes tienen en común tres operaciones básicas, que aplican sobre los elementos para transformarlos [5].

- *Traslación*: Desplaza un punto $p = (p_x, p_y, p_z)$ a otra posición $p' = (p_x, p_y, p_z)$. En la forma simple, es decir, en un sistema donde solo hay traslaciones se representa con un vector, de la forma $t = (t_x, t_y, t_z)$ dando lugar a $p' = p + t$, lo que es igual a $p' = (p_x + t_x, p_y + t_y, p_z + t_z)$. Sin embargo, cuando hay otras operaciones, se

recurre a la notación mediante matriz 3.1.

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.1}$$

Y el punto resultante queda representado en la ecuación 3.2.

$$p' = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{pmatrix} \tag{3.2}$$

- *Rotación*: Rota un punto Φ grados alrededor de un eje de coordenadas. Se representa mediante $R_x(\Phi)$, $R_y(\Theta)$ y $R_z(\Psi)$, que realizan una rotación sobre los ejes X, Y y Z, respectivamente. Los valores de estas vienen dados por las ecuaciones 3.3 - 3.5.

$$R_x(\Phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\Phi) & -\sin(\Phi) & 0 \\ 0 & \sin(\Phi) & \cos(\Phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

$$R_y(\Theta) = \begin{pmatrix} \cos(\Theta) & 0 & \sin(\Theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\Theta) & 0 & \cos(& \\ Theta) & 0 & & \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.4}$$

$$R_z(\Psi) = \begin{pmatrix} \cos(\Psi) & -\sin(\Psi) & 0 & 0 \\ \sin(\Psi) & \cos(\Psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.5}$$

Multiplicar una de estas matrices con un punto p, en este orden, creará una rotación sobre el eje correspondiente. Hay que tener cuenta que esta rotación se realiza sobre el eje de coordenadas que pasa por el origen. Si se desease realizar esta operación sobre el eje de coordenadas x, cuyo origen p sea un punto diferente al $(0,0)$ se debería trasladar la figura que se desea rotar de tal manera que el punto p, coincidiese con el origen, es decir, aplicar $T(-p)$. En este punto se hará la rotación $R_x(\Phi)$ y finalmente, se trasladaría la figura a su posición origina, $T(p)$.

- *Escalado*: Es una transformación que agranda o empequeñece las figuras. Queda representado por la ecuación 3.6.

$$S = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.6}$$

Si todos lo valores de S no son iguales, de llama escalado no uniforme, y la figura quedará deformada.

Cuando una o más de estas operaciones se quieren aplicar, se multiplican en orden inverso y al final se multiplica por el punto o puntos a transformar. Esto es, si se quiere rotar una figura y después trasladarla, la matriz resultante es la ecuación 3.7 y no la 3.8.

$$p' = T(t_x, t_y, t_z) * R_x(\Phi) * p \tag{3.7}$$

$$p' = R_x(\Phi) * T(t_x, t_y, t_z) * p \tag{3.8}$$

Estas son las transformaciones sobre un espacio tridimensional, sin embargo, a lo largo de este capítulo se va a tratar el tema de la creación de imágenes panorámicas, en las cuales, normalmente, solo existen dos dimensiones. Esto supone que el eje Z desaparece, simplificando mucho las operaciones. La traslación y el escalado solo tendrán dos variables de transformación, la X y la Y, mientras que solo habrá una transformación de rotación. Las nuevas ecuaciones son las siguientes.

$$T = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \tag{3.9}$$

$$R(\Phi) = \begin{pmatrix} \cos(\Phi) & -\sin(\Phi) & 0 \\ \sin(\Phi) & \cos(\Phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.10}$$

$$S = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.11}$$

3.2 Algoritmo genético

3.2.1 Presentación del problema

Se parte de dos imágenes distintas de las misma escena, las cuales tienen una proporción en común y se quiere formar una panorámica. Mientras que una funciona de pivote, a la otra se le aplican transformaciones para calcular su nueva posición. Aunque hay otras formas de hacerlo, se llevará a cabo mediante un algoritmo genético.

El código que se puede ver en el anexo A, no tiene rotaciones ni escalado, para que el lector pueda entender fácilmente como un algoritmo genético funciona, sin tener demasiadas interferencias provocadas por los temas que tienen que ver con la implementación y manipulación de los elementos gráficos. Hay que recordar que el objetivo de este Trabajo de Final de Grado (TFG) es explicar como funcionan los GA y como aplicarlos en diversas situaciones, el uso de la visión por computador es solo un ejemplo. Sin embargo, si que se explicará que cambios se deberían llevar a cabo si se quisiese tratar imágenes en 3D, para mostrar que la potencia del algoritmo no varia con la introducción de una dimensión más.

3.2.2 Creación de la población

Representación de los cromosomas

Dependiendo de las transformaciones que se pueden aplicar al problema, y si el espacio de búsqueda es en 2D o en 3D, se tendrán más genes o menos. Si solo se dispone de dos dimensiones y solo se pueden aplicar traslaciones, con dos genes que representen el valor X e Y de la traslación bastaría. Si se quisiesen aplicar rotaciones se necesitaría otra variable ϕ que representase el ángulo de rotación. Si se sumase el escalado y una tercera dimensión, se tendrían un total de 9 genes t_x, t_y, t_y, Φ, Θ, Ψ, s_x, s_y y s_z [6].

Inicialización de la población

La inicialización de estas variables son completamente al azar, pero siempre estará entre lo valores máximos permitidos. Esto quiere decir, que si la imagen mide 2000x3000, la X estará entre -2000 y 2000, y la Y entre -3000 y 3000.

3.2.3 Cálculo del "fitness"

Dado que las imágenes se trasladan en el espacio, lo primero es explicar como se calcula que píxel (o vóxel si se trata de un espacio de tres dimensiones) de una imagen se sobrepone con que píxel de la otra. Aunque parezca un trabajo laborioso es tan fácil como aplicar las transformaciones a cada punto. Si se tiene P_x, un píxel de la imagen X, y se desea calcular que píxel P_y es el correspondiente en la imagen Y:

$$P_y = M * P_x = T(t_x, t_y, t_z) * R_z(\Psi) * R_y(\Theta) * R_x(\Phi) * S(s_x, s_y, s_z) * P_x \tag{3.12}$$

Dado que solo se tiene traslaciones, se debe sumar a la X y a la Y sus respectivos valores de traslación.

$$P_y = T + P_x = T(t_x, t_y) + P(p_x, p_y) \tag{3.13}$$

Este píxel P_y puede dar fuera de los margenes de la imagen y entonces será ignorado a la hora de comprobar el "fitness". Existen diversos métodos para obtener la similitud de dos imágenes, que se clasifican en aquellos basados en intensidad y aquellos basados en características. Mientras que los primeros comparan colores o intensidades en los píxeles, los segundos buscan correspondencias entre las imágenes, como contornos o lineas. Independientemente de si el espacio de búsqueda es de dos o tres dimensiones, los algoritmos de similitud se pueden extrapolar, cambiando la ecuación que se aplica a la hora de calcular los píxeles o vóxeles correspondientes. Para este TFG se decidió probar con los basados en intensidades.

El primer algoritmo de "fitness" que se programó comparaba uno a uno todos los píxeles que se solapaban, cada uno con su correspondiente en la otra imagen. Además de lento, no era muy efectivo, ya que solamente cuando coincidían plenamente daba una similitud elevada. Lo siguiente que se intentó fue calcular la media de intensidad que había en la fracción de las imágenes que se solapaban. Este método seguía tardando demasiado tiempo, pero era mucho más eficaz que el anterior.

Se decidió probar con un algoritmo probabilístico para reducir el tiempo de cálculo. Dado que se había comprobado que comparar píxel a píxel no era efectivo, pero si lo

era comprobando una media, se comparaba la media de intensidades de zonas de 5x5 píxeles, escogidos al azar. Si esta comprobación se hace repetidas veces para una misma posición de la imagen, el resultado es muy positivo. A diferencia de lo que se puede pensar, las comparaciones no se debería hacer con media de colores si las imágenes no son sintéticas, ya que cualquier cambio de luz podría invalidar este proceso. Por eso mismo, antes de realizar la media los valores *RGB* de los píxeles se pasan a blanco y negro, con la siguiente formula.

$$I = R * 0,3 + G * 0,59 + B * 0,11 \tag{3.14}$$

Si se hace la media entre los 25 píxeles de las cada imagen como se indica en la siguiente ecuación,

$$Avg_1 = \frac{\sum_{i=1}^{5}\sum_{j=1}^{5} I_{(i,j)}}{25} \tag{3.15}$$

y posteriormente se calcula la diferencia entre ellos de la forma

$$PointFitness = (255 - \mid Avg_1 - Avg_2 \mid)/255 \tag{3.16}$$

se consigue un valor entre 0 y 1 para la similitud entre puntos, donde 1 es completa y 0 es no parecido. Como esto se debe repetir X veces, el valor quedará entre 0 y X o se puede normalizar dividiendo entre el número de veces a realizar la comparación.

3.2.4 Selección

Cualquiera de los métodos comentado en la sección 2.1.2 es válido para ser implementado en el registro de imágenes. Sin embargo, los que ponemos ver en la sección A.3 son los basados en la ruleta, que favorecen la diversidad entre los genes de los cromosomas.

Cuando los otros eran usados, dado que unicamente se tenían dos variables, se llegaba muy rápido a una situación donde todos los individuos tenían los mismos o casi los mismos valores para X y para Y. Si las nueves variables anteriormente comentadas formasen un cromosoma, los otros métodos no se quedarían tan atrás en cuanto a eficiencia.

3.2.5 "Crossover"

Cuando los dos padres se han seleccionado, se debe aplicar la recombinación. La existencia de solo dos genes codificados como valores numéricos no deja mucha elección. Los dos cromosomas escogidos se intercambia entre ellos los valores de la X y de la Y, creando un nuevo individuo con la X del padre 1 y la Y del padre 2 y, otro, con la X del padre 2 y la Y del 1. De nuevo, si las nueve variables estuviesen en juego, se podrían seleccionar al azar de que padre hereda el gen.

3.2.6 Elitismo

Dos versiones de elitismo se aplican en este problema. Primero, se guarda el mejor cromosoma encontrado , ya que las mutaciones o reproducciones pueden hacer perder el mejor en la generación actual. Dado que se ordenan los individuos para la selección, se aprovecha para obtener el mejor. Por otra parte, 12 de los mejores individuos se pasan

directamente a la siguiente generación. Estos individuos puedes ser seleccionados posteriormente para la reproducción.

3.2.7 Mutación

Aplicar una mutación sobre valores numéricos es más intuitivo que cambiar 0's por 1's en los genes de un cromosoma, se efectúa una suma o una resta con un valor al azar sobre ellos [7].

Al principio, el valor de esta suma iba de -50 a 50, pero esto provocaba que, eventualmente, no se explorase cierta parte del espacio de búsqueda y se necesitaban más generaciones o una población más grande con tal de encontrar una solución. Se decidió aumentar el rango de este valor de -500 a 500. Aunque ahora si que se exploraban todo el espacio de búsqueda, la solución no era la óptima, y las imágenes de la panorámica generada estaban una media de 15 píxeles desplazados de la solución idónea.

Se decidió modificar un poco el algoritmo, haciendo que al principio, el valor de esta mutación fuese muy grande, permitiendo le mover la imagen por todo el espacio. Una vez se alcanzaba la mitad de las generaciones establecidas, este rango en la mutación se disminuiría drásticamente, favoreciendo le encontrar la solución óptima [8]. En este caso, el desplazamiento de píxeles entre las imágenes respecto es de solo 2.

3.2.8 Criterio de parada

Aunque al principio se necesitaban mas de 3000 generaciones y casi 5 minutos de ejecución para conseguir un resultado aceptable, ahora, solo 500 generaciones y unos 15 segundos, bastan para obtener la panorámica casi sin distorsión. Muchas veces la solución es encontrada antes, por ello se ha ideado un criterio de parada distinto al número de generadas ejecutas, sino basado en la similitud del mejor individuo encontrado.

Dado que en el cálculo de "fitness" hemos normalizado el valor, este estará entre 0 y 1. Debido a problemas de redondeo de valores decimales, aunque las dos imágenes este perfectamente alineadas no dará el valor máximo. Se ha establecido un valor muy próximo, 0.995, que asegura que la imagen tenga como máximo, dependiendo de la fotografía, 1 ó 2 píxeles de desplazamiento.

3.2.9 Juego de pruebas

La teoría siempre es muy bonita, y aunque el código se puede probar y manipular, pocos de los lectores tendrán tiempo o ganas para hacerlo, así que ahora se enseñará los resultados de una ejecución, para demostrar la potencia del algoritmo.

La figura 3.1 muestra las dos fotografías tomadas, que tiene una parte del puente en común. Como ya se ha comentado la ejecución hace un máximo de 500 generaciones, y acepta como válida la imagen cuando tiene un "fitness" mayor que 0.995. El proceso ha tardado 7 segundos y realizó 82 generaciones. Dado que el algoritmo usa el azar, no siempre tarda lo mismo, sin embargo no suele tomar más de 15 segundos. Teniendo en cuenta que las imágenes tienen un tamaño de 3000x2000 píxeles, es bastante rápido.

Podemos ver el resultado en la figura 3.2

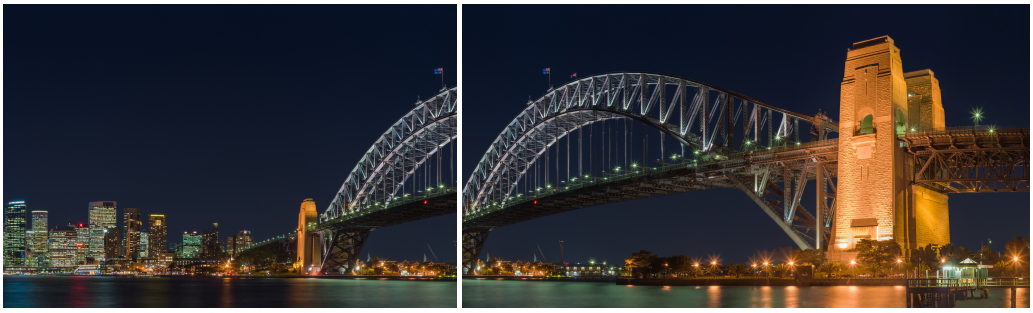

Figura 3.1: Partes de la panorámica de Sidney

Figura 3.2: Panorámica completa Sidney

Sin embargo, aunque a simple vista parece perfecta, si hacemos zoom y miramos los puntos donde ambas imágenes se juntan, podemos ver que hay una pequeña diferencia, que no suele superar los 3 píxeles.

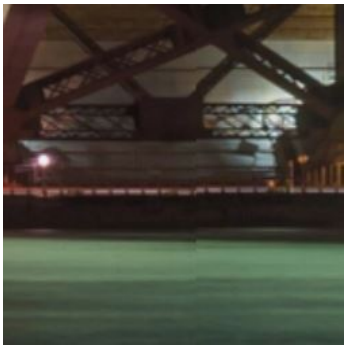

Figura 3.3: Zoom en el error de panorámica

3.3 Resumen

En este capítulo se ha visto como crear una fotografía panorámica a partir de dos imágenes de la misma escena mediante los algortimos genético. El GA se ha implementado de la siguiente forma:

- *Representación del problema*: Cada cromosoma esta formado por una X y una Y, que indica la posición actual de la imagen.
- *"Fitness"*: Se compara, de forma probabilistica, los píxeles de regiones superpuestas de las imágenes.
- *Selección de progenitores*: Se hace uso de la ruleta del ranking lineal, que establece la probabilidad de selección según la posición que ocupa en un ranking según el "fitness" de cada individuo, donde aquel con un valor más alto tiene más posibilidades de ser seleccionado.
- *"Crossover"*: Se usa una variación de recombinación en un punto. Se crean nuevos individuos con la X y la Y de los padres, proviniendo cada una de uno distinto.
- *Mutación*: Se suma un valor al azar a la X y a la Y. Cuando se ha llegado a la mitad de las generaciones máximas establecidas, este valor se reduce drásticamente, para intentar encontrar una solución óptima.

3.3.1 Mejoras

Aunque el resultado conseguido por este algoritmo es una panorámica muy conseguida, tiene una pequeña frontera donde, si se hace zoom, se aprecia un desplazamiento de píxeles. Dado que la panorámica tiene un tamaño de 6000x2000, un par de píxeles es casi despreciable.

Igual que la mayoría de aplicación de generación de panorámicas, el desplazamiento de píxeles se puede tratar con una fusión de píxeles, que reduce el impacto visual que produce. Si este desplazamiento es muy grande, el fusionado no es muy eficaz. Otra técnica más costosa es comparar píxel a píxel la "frontera", para intentar encontrar contornos comunes, o diferencia de colores. De esta forma se puede obtener los píxeles de distancia entre las imágenes. Dado que en este caso, el desplazamiento es muy pequeño, la fusión sería suficiente, para no aumentar el tiempo total de generación de panorámicas.

CAPÍTULO 4

ALGORITMOS GENÉTICOS APLICADO A LA CRIPTOLOGÍA

4.1 Criptología

Se llama criptología a la disciplina que se encarga de codificar, mediante lo que se denomina una clave, la información de forma que sea imposible o muy difícil de entender para aquellos quienes no son los destinatarios del mismo. Se divide en dos aspectos, la criptografía, que se encarga que encriptar y desencriptar el mensaje teniendo la clave, y el criptoanalisis, que se encarga de desencriptar el mensaje transmitido sin tener en posesión la clave de encriptado.

Aunque los algortimos genéticos se han usado en los dos campos, aquí se va a entrar en detalle en el proceso de criptoanalisis. Lo que el algoritmo debe hacer es encontrar la clave que se ha usado para encriptarlo.

Aunque hay muchos método de encriptado, algunos de los cuales no se pueden resolver con algoritmos genéticos, en este documento es el Cifrado de Substitución Monoalfabetico (MSC). Este método substituye cada letra del alfabeto en el texto siempre por otra letra o símbolo. La correspondencia entre letra del texto original y del encriptado es la antes mencionada clave, y es única en ambos sentidos. Otro métodos que han sido probados con algortimos genéticos [9] son los siguientes:

- *Cifrado de substitución polialfbética*: Funciona de forma similar a la MSC, sin embargo, la clave cambia cada cierto tiempo, ya sea cada número de letras o al leer un símbolo como por ejemplo un punto, a lo largo del texto.
- *Cifrado de permutación*: Los caracteres no son substituidos por otros, sino que cambian su posición en la palabra, frase o texto.
- *Cifrado de transposición*: Se escribe el texto en forma de "rectángulo", de la forma que muestra el imagen de la izquierda de la figura 4.1. Todas las palabras se ordenan siguiendo el orden alfabético de la primera palabra del rectángulo,

quedando como resultado la segunda imagen de la misma figura. El texto “Ejemplo de prueba para cifrado de transposición” queda transformado en “Eejlmop dp eerbua apcr iafardd eo tarpnosscinio”. Debido a que funcionan como si de columnas se tratase, también se llama permutación columnar.

1	*3*	*2*	*5*	*7*	*4*	*6*
E	J	E	M	P	L	O
D	E	P	R	U	E	B
A	P	A	R	A	C	I
F	R	A	D	O	D	E
T	R	A	N	S	P	O
S	I	C	I	O	N	

=>

1	*2*	*3*	*4*	*5*	*6*	*7*
E	E	J	L	M	O	P
D	P	E	E	R	B	U
A	A	P	C	R	I	A
F	A	R	D	D	E	O
T	A	R	P	N	O	S
S	C	I	N	I	O	

Figura 4.1: Cifrado de transposición

4.2 Algoritmo genético

4.2.1 Planteamiento

A partir un texto cifrado se debe obtener la clave con la cual se ha descifrado, para aplicarla y obtener el texto original. La técnica de cifrado que se quiere romper es la MSC. El texto que se quiere descifrar es una parte de “El Quijote”:

> En efecto, rematado ya su juicio, vino a dar en el más extraño pensamiento que jamás dio loco en el mundo, y fue que le pareció convenible y necesario, así para el aumento de su honra, como para el servicio de su república, hacerse caballero andante, e irse por todo el mundo con sus armas y caballo a buscar las aventuras, y a ejercitarse en todo aquello que él había leído, que los caballeros andantes se ejercitaban, deshaciendo todo género de agravio, y poniéndose en ocasiones y peligros, donde acabándolos, cobrase eterno nombre y fama.

Este texto, será encriptado con una clave creada al azar, que cambiará en cada ejecución del problema.

4.2.2 Creación de la población

Representación de los cromosomas

La forma más obvia para realizar la representación de los datos es una “array” indexado por enteros (dado que C♯ no permite indexar por caracteres), donde cada entero representa un carácter, y contiene la substitución que se debe hacer en el texto. La correspondencia entre caracteres e indices es, obviamente, el orden alfabético. De esta forma, la figura 4.2 muestra un ejemplo de como sería la representación de una clave.

Cada una de las casillas del “array” que representa la correspondencia entre dos letras es un gen.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
K	E	B	Z	H	P	U	L	V	A	W	O	I	D

15	16	17	18	19	20	21	22	23	24	25	26
T	Y	G	N	J	S	R	C	M	Q	X	F

Figura 4.2: Ejemple de clave

Inicialización de la población

Dado que no se pueden repetir elementos en el "array", se crea uno que tiene todas las letras del alfabeto en orden alfabético. Posteriormente, se procede al desordenamiento, al azar. De esta forma se asegura que se creará una clave donde los elementos no se repetirán, pero que cada uno será diferente.

4.2.3 Cálculo del "fitness"

Saber si el texto desencriptado es correcto o no no es tarea fácil, y mucho menos, siendo dos textos incorrectos, saber cual es "mejor" que otro.

El primer intento, que aunque suena bastante bueno fue un fracaso, consistía en tener una tabla con las frecuencias de aparición de cada letra. Al desencriptar el texto, esta probabilidad se sumaba y aquel que tuviese el resultado más alto, era considerado el mejor. Sin embargo, lo único que esto conseguía era frases sin sentido alguno. Aunque con las vocales acertaba alguna que otra, no pasaba lo mismo con las consonantes.

La tabla de probabilidades de las letras era sin duda el camino a seguir. Sin embargo, lo erróneo era tomarlas individualmente. Se implementó una tabla de frecuencias de N-gramas [10], es decir, todas las posibles combinaciones de N letras de la lengua. Al hacerlo con 2-gramas se podía observar como el número de letras acertadas mejoraba bastante, pero aún así entender el texto costaba demasiado como para llamarlo un éxito. Se decidió probar con 3-gramas, juntamente con 2-gramas. El número de letras acertado era superior al 70 %. Con esta mejora, lo más lógico era pensar que si se sumaban 4-gramas el texto sería muchísimo mejor. No solo no mejoraba casi nada el desencriptado, si no que encima añadía demasiado coste computacional, incrementando el tiempo necesario exageradamente.

Aunque el resultado era satisfactorio, se podrían realizar algunas mejoras. Se implementó una penalización cuando el n-grama no existía en la lengua, es decir, su valor de frecuencia era cero. También se añadió un peso a lo que afectaba al "fitness' los 2-gramas y 3-gramas. Por último, se paso a considerar elementos como comas, puntos, y espacios como una misma letra, de forma que simulase el inicio y final de palabra. Todas estas mejoraron el número de letras acertadas, haciendo que el texto fuese mucho más fácil de desencriptar.

4.2.4 Selección

Dado que los métodos de selección más comúnmente usados son selección de la ruleta y ranking lineal y que han demostrado su eficiencia, se ha decidido reaprovecharlos.

4.2.5 "Crossover"

Ninguno de los anteriores métodos de reproducción explicados en el punto 2.1.2 se puede aplicar directamente, ya que crearía duplicidades de caracteres en la tabla. Para evitar que esto sucediese se ha implementado el siguiente método [11]. Los ejemplos puestos tiene solo 8 letras, para facilitar la lectura.

- Se selecciona un gen aleatorio del padre 1, imaginemos que es la 'd', que ocupa la posición 6, y se inserta en el hijo 1. Figura 4.3

1	2	3	4	5	6	7	8
C	E	G	F	A	D	H	B

Padre 1

1	2	3	4	5	6	7	8
F	G	D	B	A	E	H	C

Padre 2

1	2	3	4	5	6	7	8
					D		

Hijo 1

Figura 4.3: Paso 1 recombinación criptología

- La letra que ocupa la posición recién ocupada, la 6 en el padre 2, pongamos que es la 'e', no puede ser seleccionada para pasar al hijo, ya que este gen esta "ocupado". Figura 4.4

1	2	3	4	5	6	7	8
C	E	G	F	A	D	H	B

Padre 1

1	2	3	4	5	6	7	8
F	G	D	B	A	E	H	C

Padre 2

1	2	3	4	5	6	7	8
					D		

Hijo 1

Figura 4.4: Paso 2 recombinación criptología

- Si queremos que todas las letras estén el nuevo hijo, este tiene que heredar la letra 'e' del padre 1. Figura 4.5
- Tras repetir estos pasos varias veces, se llega a un punto donde, en el tercer paso, el hijo ya tiene esta letra. Figura 4.6
- En este punto, se heredan todas las letras en los genes no ocupados del padre 2.Figura 4.7

1	2	3	4	5	6	7	8
C	E	G	F	A	D	H	B

Padre 1

1	2	3	4	5	6	7	8
F	G	D	B	A	E	H	C

Padre 2

1	2	3	4	5	6	7	8
	E				D		

Hijo 1

Figura 4.5: Paso 3 recombinación criptología

1	2	3	4	5	6	7	8
C	E	G	F	A	D	H	B

Padre 1

1	2	3	4	5	6	7	8
F	G	D	B	A	E	H	C

Padre 2

1	2	3	4	5	6	7	8
	E	G			D		

Hijo 1

Figura 4.6: Paso 4 recombinación criptología

1	2	3	4	5	6	7	8
C	E	G	F	A	D	H	B

Padre 1

1	2	3	4	5	6	7	8
F	G	D	B	A	E	H	C

Padre 2

1	2	3	4	5	6	7	8
F	E	G	B	A	D	H	C

Hijo 1

Figura 4.7: Paso 5 recombinación criptología

- Se repite los pasos para el otro hijo.

4.2.6 Elitismo

De la misma forma que el registro de imágenes, dos versiones de elitismo se aplican en este problema. Se guarda el mejor cromosoma encontrado y 12 de los mejores individuos se pasan directamente a la siguiente generación. Estos individuos puedes ser seleccionados posteriormente para la reproducción.

4.2.7 No “crossover”

Otra técnica de no “crossover” no comentada anteriormente que se ha utilizado en este problema, es la introducción de nuevos cromosomas en cada generación. El principal problema de este algoritmo era la convergencia hacia una misma clave, provocando que se quedase atascado en estados mejorables. Aunque el resultado no estaba mal, era muy difícil que, por muchas generaciones que se estableciesen, se llegase a descubrir un cromosoma mejor.

Se podría haber resuelto de forma similar como se hizo con el registro de imágenes, modificando el método de mutación, haciendo que alcanzadas cierto número de generaciones, más genes fuesen mutados. Sin embargo, se quiso probar este nuevo método.

La introducción de nuevos cromosomas, permite una diversidad en la población, y puede resultar crucial para encontrar una clave mejor. Este método es más seguro que la mutación, ya que mientras la segunda es totalmente al azar y puede resultar en un cromosoma peor que el existente, la introducción de un nuevo cromosoma en la población muy malo provocará que su probabilidad de selección sea bajo y desaparezca, pero si por el contrario es un cromosoma muy bueno será seleccionado para la reproducción.

4.2.8 Mutación

Las letras en la clave no se pueden repetir, así no se pueden alterar individualmente cada gen. Hay varias formas de mutar un individuo.

La primera es intercambiar un rango de genes consecutivos con otra posición. La que se utiliza en el código del anexo B es el intercambio de un número al azar de genes. Este método, aunque es más sencillo, ha demostrado ser un poco mejor que el anterior, permitiendo acercarse más a la solución óptima.

4.2.9 Criterio de parada

Normalmente más de 1500 generaciones son necesarias para alcanzar una clave de desencriptado que nos permita entender el texto sin dificultades. De la misma forma que pasa con todos lo GA, la solución puede ser encontrada antes de tiempo, por eso es necesario un criterio de parada.

Dado que el fitness era una suma de probabilidades, daba un valor no incluido entre el 0 y el 1, así que se decidió normalizarlo. Para ello, se supuso un máximo de “fitness”, que era el resultado de la suma de que todas los n-gramas fuesen el mejor. Aunque esto es imposible, al dividir el “fitness’ por el resultado de esta suma, el valor quedaba normalizado. Se estableció como criterio de parada el valor de “fitness’ una vez normalizado como 0.25.

4.2.10 Juego de pruebas

La figura 4.2.10 muestra una ejecución del problema en el cual, con solo 750 ejecuciones se ha obtenido un resultado muy bueno, con un “fitness” de 0.255. Podemos observar los tres textos: el original, el encriptado y el desencriptado. Podemos observar que entender el texto encriptado es imposible sin ayuda de la clave. El obtenido por el

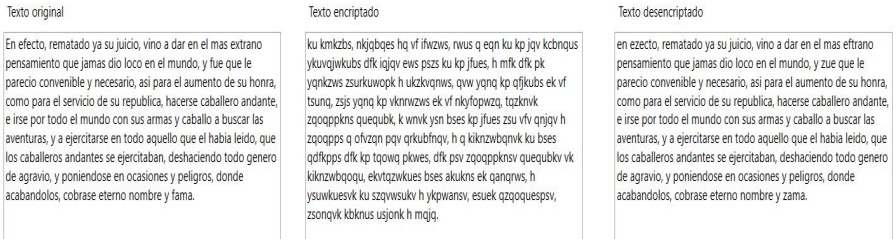

Figura 4.8: Ejemplo de GA en criptología

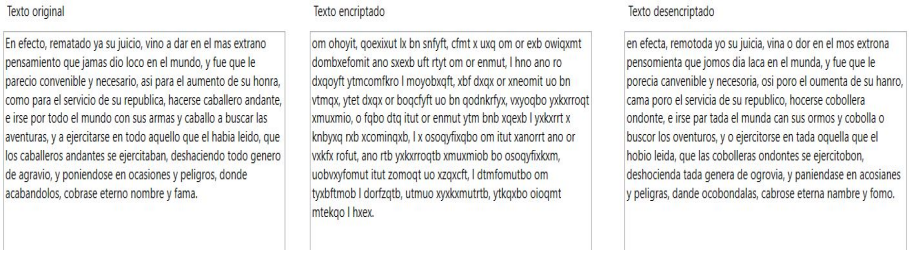

Figura 4.9: Otro ejemplo de GA en criptología

algoritmo genético no es perfecto, ya que hay 3 letras mal colocadas, la f, x y la z. La aparición de estas en el texto son muy escasas, aparecen 3, 1 y 0 veces, respectivamente, así que el texto es perfectamente entendible.

Sin embargo, la mayoría de las veces no se obtiene un texto tan perfecto, aunque si que igualmente entendible. La figura 4.2.10 muestra una ejecución donde las 1500 generaciones no fueron suficientes y se quedó en un "fitness" de 2,40.

Aunque la mayoría de las letras están bien colocadas, la "a" y la "o" están intercambiadas, dificultando mucho la lectura del texto.

4.3 Resumen

En este capítulo se ha visto como encontrar la clave de encriptado de un texto mediante los algortimos genéticos. El GA se ha implementado de la siguiente forma:

- *Representación del problema*: Cada cromosoma esta formado por un "array" que representa la clave de cifrado, indicando las substituciones de letras que se deben realizar en el texto.
- *"Fitness"*: Se suman las probabilidades de los n-gramas (conjunto de n letras) que forman el texto una vez desencriptado. La probabilidad de cada n-grama se obtienen haciendo un estudio previo de un texto de a misma lengua.
- *Selección de progenitores*: Se hace uso de la ruleta del ranking lineal, que establece la probabilidad de selección según la posición que ocupa en un ranking según el "fitness" de cada individuo, donde aquel con un valor más alto tiene más posibilidades de ser seleccionado.

- *"Crossover"*: Debido a que no se pueden repetir valores en una clave, se ha tenido que modificar el método de selección de un solo punto, creando una ciclo que cambiase todos los genes que provocarián dicha repetición.

- *Mutación*: Se intercambia entre uno y cinco genes por otros dentro del cromosoma, alterando el orden original.

4.3.1 Mejoras

El resultado de los textos descifrados no es perfecto, pero el número de letras acertadas es más de un 80 %, lo que hace que el texto sea fácil de entender. Aunque se aumenten las generaciones a realizar, el resultado, normalmente, se queda estancado en este punto. Aumentar el tamaño de la población es mucho más eficiente, mejorando los resultado a costa de mucho más tiempo.

Como ya se ha comentado varias veces a lo largo de este documento, un algoritmo genético no suele encontrar la solución óptima, pero si que se acerca mucho. En este punto, la mejora de la calidad descifrado tendría que ser un algoritmo simple fuera del propio GA que se encargase de detectar que letras no están bien posicionadas en las claves. Dado que las palabras están casi formadas seria mucho menos costoso que detectar directamente la posición de la letras sobre el texto encriptado.

CAPÍTULO 5

CONCLUSIONES

A lo largo del documento se ha hecho una introducción a los algoritmos genéticos como un método de optimización, explicando sus elementos y operadores.

En el primer capítulo se han presentado la inteligencia artificial y su uso en la resolución de problemas, explicando que es un estado, una búsqueda y demás elementos básicos, necesarios para la posterior explicación de los algoritmos genéticos.

En el segundo capítulo se han presentado los GA. Basados en la ley de la selección natural, la analogía es más que clara. En la naturaleza, las especies evolucionan con cada generación, donde solo aquellos individuos más aptos para la supervivencia son capaces de transmitir sus genes a sus descendientes mediante la reproducción. Además, un toque de azar, llamado mutación, permite incorporar genes o características nunca antes presentes en una especie.

Los algoritmos genéticos funcionan de una forma muy parecida. Se empieza por crear una población de estados (individuos) obtenidos completamente al azar. A cada uno de estos, se les asigna una puntuación ("fitness") que, mientras que en la naturaleza equivale a la capacidad de los individuos a sobrevivir, en los GA indica que tan bueno es este estado para la solución del problema. Aquellos con una puntuación más alta, es decir lo mas aptos, tendrán más posibilidades de ser seleccionados para transmitir sus genes a la siguiente generación. En estos algoritmos, la mutación también se puede aplicar, creando estados que no se podrían, o sería muy difícil, crear unicamente a partir de la reproducción.

De esta forma, cada generación irá acumulando las mejores cualidades y características que se han descubierto. Si, tanto el algoritmo como el problema han sido bien estudiados y diseñados, la población convergerá hacia una solución óptima.

En este mismo capítulo, se ha podido ver, con un pequeño ejemplo, como se aplican todos los elementos explicados. Sin embargo, es en los siguientes capítulos donde los algoritmos genéticos se aplican a problemas "reales".

En el tercer capítulo se ha utilizado este método de resolución en el registro de imágenes. La finalidad era crear una imagen panorámica a partir de dos fotografías distintas de la misma escena que solo tuviesen una parte en común. Utilizando como

genes de los cromosomas una variable X y otra variable Y, con valores enteros, se ha conseguido un resultado muy bueno, necesitando relativamente poco tiempo de ejecución. Además, las operaciones de "fitness", "crossover" y mutación no son mucho más complejas que en el ejemplo del capítulo 2. Por otra parte, se ha usado la ruleta para la selección de progenitores y se ha implementado un método de elitismo, que mejora enormemente la eficiencia de un algoritmo genético.

Finalmente, en el cuarto capítulo, se ha aplicado esta técnica en la criptología, en concreto, el desencriptado de un texto simple. Aquí, ya se han presentado unas operaciones un poco más complejas. Los genes son una representación de la clave de desencriptado, en un "array" de caracteres. Los operadores se han estudiado y son un poco más complejos, ya que se debía evitar la repetición de letras en la clave. Además, se ha implementado una nueva técnica de "no-crossover", la introducción de elementos nuevos en las poblaciones, para evitar la rápida convergencia de la población.

5.1 Ventajas y desventajas

Una de las principales diferencias sobre la mayoría de algoritmos de optimización, y que a su vez lo hace tan útil, es el tratamiento de la búsqueda en paralelo. Mientras que los otros parten de uno o varios puntos y buscan secuencialmente la mejor solución a partir de cada uno de ellos, los GA parten de varios puntos al azar, y se comparten los avances. Esto permite evitar caer en óptimos locales, problema con los que pecan otros métodos.

Otra clara ventaja que tienen este método es que permite reducir los elementos a comprobar en el espacio de búsqueda. De esta forma, es capaz de solucionar más rápidamente que otros algoritmos problemas con una gran cantidad de estados posibles. Esto solo sucede cuando los operadores de selección, reproducción y de cálculo de "fitness" están correctamente implementados. Si no es así, puede que la población converja demasiado deprisa, sin dar tiempo a explorar otras partes del espacio de búsqueda o, por el contrario no converger y estancarse en los mismos estado generación tras generación.

Una de las principales desventajas es que la idea se basa en reglas probabilisticas, lo que provoca que sea muy complicado que encuentre la solución perfecta, sino que, como se ha visto en los ejemplos, se queda próxima a ella sin ser capaz de alcanzarla. A pesar de ello, numerosos estudios han demostrado que estas soluciones son más que aceptables [12].

Aunque permiten solucionar muchos problemas de distinta área e índole, suelen quedar por detrás de técnicas especializadas para resolver un determinado problema.

Todas estas ventajas y desventajas comentadas en este punto, se han podido observar con los ejemplos solucionados. El registro de imágenes y la criptología, de dos campos completamente distintos, han podido ser solucionados con el mismo algoritmo, cambio una sola clase (la clase cromosoma), sin caer en óptimos locales y de una forma rápida, aunque sin poder alcanzar la solución óptima. Por otra parte, esta claro que otras técnicas más especializadas en cada uno de ellos pueden solucionar estos problemas con una precisión mayor.

APÉNDICE A

CÓDIGO ALGORITMOS GENÉTICOS EN REGISTRO DE IMÁGENES

A.1 Helper.cs

Clase que tiene todas las variables que el usuario puede modificar para alterar el comportamiento del algoritmo. Están separadas para facilitar la localización de las mismas.

```
1  using System;
2  using System.Drawing;
3  using System.Windows.Media.Imaging;
4
5  namespace panoramic {
6  class Helper {
7
8    //Variables de propias del GA
9
10   //Tamaño de la población
11   public static int populationSize = 64;
12   //Probabilidad de que dos individuos se reproduzcan (rango 0-1)
13   public static double crossoverRate = 0.85;
14   //Probabilidad de mutación de cada individuo (rango 0-1)
15   public static double mutationRate = 0.33;
16   //Rango de la suma de la mutación
17   public static int mutationRange = 500;
18   //Número de individuos elitistas
19   public static int elitismRange = 16;
20   //Indica si se usa ruleta por fitness o ranking (true/false, respectivamente)
21   public static bool useFitnessRoulette = false;
22   //Fitness aceptado como solución
23   public static double threshold = 0.995;
24   //Número de generaciones que se hará si no se encuentra el fitness antes
25   public static int numberOfGenerations = 500;
26
27
28   //Rango que deben tener en común para ser válidas
```

```
29    public static double commonRange = 20;
30    //Puntos a comparar para calcular el fitnes
31    public static int pointsToCompare = 40;
32
33    //Clase de c# para obtener número aleatorios
34    public static Random rnd = new Random();
35    //Url de las imagenes
36    public static String urlImage = "../../images/Sidney2.jpg";
37    public static String urlImage2 = "../../images/Sidney1.jpg";
38    public static Uri uri = new Uri(Helper.urlImage, UriKind.Relative);
39    public static Uri uri2 = new Uri(Helper.urlImage2, UriKind.Relative);
40
41    //Imagen para poder obtener los colores de los pixeles
42    public static Bitmap bitmap = new Bitmap(Helper.urlImage);
43    public static Bitmap bitmap2 = new Bitmap(Helper.urlImage2);
44    //Imagen para wpf (mostrar en interfaz gráfica
45    public static BitmapImage bitmapImage = new BitmapImage(uri);
46    public static BitmapImage bitmapImage2 = new BitmapImage(uri2);
47
48
49    }
50 }
```

A.2 MyChromosome.cs

Clase que representa los datos de los cromosomas, así como las operaciones que se realizan sobre estas. Inicializa, reproduce, muta y calcula el "fitness" de los individuos.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace panoramic {
8    class MyChromosome {
9      //Posición X de la imagen
10     private int xValue;
11     //Posición Y de la imagen
12     private int yValue;
13     //Fitness del individuo
14     private double fitness;
15
16     //Crea un nuevo individuo asignandole una X y una Y al azar
17     public MyChromosome() {
18       XValue = Helper.rnd.Next(-Helper.bitmap.Width, Helper.bitmap.Width);
19       YValue = Helper.rnd.Next(-Helper.bitmap.Height, Helper.bitmap.Height);
20       Fitness = -1;
21     }
22     //Crea un individuo en base a los parametros
23     public MyChromosome(int x, int y, double f) {
24       XValue = x;
25       YValue = y;
26       Fitness = f;
27     }
28     //Getters y setters
```

```
29      public int XValue {
30        get {
31          return xValue;
32        }
33
34        set {
35          xValue = value;
36        }
37      }
38
39      public int YValue {
40        get {
41          return yValue;
42        }
43
44        set {
45          yValue = value;
46        }
47      }
48
49      public double Fitness {
50        get {
51          if (fitness == -1) {
52            calcFitness();
53          }
54          return fitness;
55        }
56
57        set {
58          fitness = value;
59        }
60      }
61      //Cambiar a completa si se desea (nada aconsejable si no quieres tardar años)
62      public void calcFitness() {
63        calcFitnessProb();
64      }
65
66      /*
67      * Calcula el fitness del individuo. Lo hace con un método probabilistico,
            cogiendo X puntos al azar. En cada punto hace la media de los
68      * Colores de los pixeles en la redonda 5x5. Con esta media compara y le da
            unos puntos
69      */
70      public void calcFitnessProb() {
71        //Si se produce error significa que las imágenes no coinciden
72        try {
73          System.Drawing.Color c;
74          System.Drawing.Color c2;
75          //Calculamos el espacio de intersección de las imágenes
76          int maxX = Math.Min((int)Helper.bitmap.Width, (int)Helper.bitmap2.Width
                + XValue);
77          int minX = Math.Max(0, XValue);
78          int maxY = Math.Min((int)Helper.bitmap.Height,
                (int)Helper.bitmap2.Height + YValue);
79          int minY = Math.Max(0, YValue);
80          //Las imágenes no coinciden
81          if (maxX < minX || maxY < minY) {
```

```
82            Fitness = 0;
83            return;
84          }
85          //Establecemos un porcentaje minimo de solapamiento
86          int xPixelsOverloap = maxX - minX;
87          int yPixelsOverloap = maxY - minY;
88          double totalPixels = xPixelsOverloap * yPixelsOverloap;
89          double percent = totalPixels / (Helper.bitmap.Width *
                Helper.bitmap.Height * 2 - totalPixels) * 100;
90
91          //Todos los valores que provocan que la imagen no se superpone quedan
                excluidos
92          if (percent < Helper.commonRange) {
93            Fitness = 0;
94            return;
95          }
96          //Bucle de comparación de los X puntos
97          int x;
98          int y;
99          Fitness = 0;
100         for (int i = 0; i < Helper.pointsToCompare; i++) {
101
102           double avgC = 0;
103           double avgC2 = 0;
104           x = Helper.rnd.Next(minX, maxX - 10);
105           y = Helper.rnd.Next(minY, maxY - 10);
106           for (int newX = x; newX < x + 5; newX++) {
107             for (int newY = y; newY < y + 5; newY++) {
108               //Obtenemos el color y lo pasamos a escala de grises
109               c = Helper.bitmap.GetPixel(newX, newY);
110               c2 = Helper.bitmap2.GetPixel(newX - XValue, newY - YValue);
111               avgC = avgC + (c.R * 0.3) + (c.G * 0.59) + (c.B * 0.11);
112               avgC2 = avgC2 + (c2.R * 0.3) + (c2.G * 0.59) + (c2.B * 0.11);
113             }
114           }
115
116           avgC = avgC / 25;
117           avgC2 = avgC2 / 25;
118           //Cada resultado ira de 0 a 1. El resultado sera entre 0 y X (número
                de puntos a comparar)
119           Fitness = Fitness + (double)(255 - Math.Abs(avgC - avgC2)) / 255;
120         }
121       } catch (Exception e) {
122         Fitness = 0;
123       }
124     }
125
126     //Intercambia la X por la Y de los dos individuos
127     public void valueSwapCrossover(MyChromosome c2) {
128       int xValueAux = XValue;
129       XValue = c2.XValue;
130       c2.XValue = xValueAux;
131     }
132     //Suma un valor a la X y otro a la Y
133     public void valueMutaton() {
134       int aux = Helper.rnd.Next(-Helper.mutationRange, Helper.mutationRange);
135       XValue = XValue + aux;
```

```
136       aux = Helper.rnd.Next(-Helper.mutationRange, Helper.mutationRange);
137       YValue = YValue + aux;
138     }
139     //Clona el individuo
140     public MyChromosome Clone() {
141       return new MyChromosome(this.XValue, this.YValue, this.Fitness);
142     }
143   }
144 }
```

A.3 MyPopulation.cs

Clase que representa la población completa que se esta estudiando, así como las operaciones necesarios para que esta evolucione. Inicializa y establece como se reproducen los individuos, así como cuales de ellos se reproducen, pasan a la siguiente población sin ningún cambio, desaperecen, mutan...

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace panoramic {
8    class MyPopulation {
9      //Población de todos los individuos
10     private MyChromosome[] currentPopulation;
11     private int size;
12     //El mejor indivuduo encontrado hasta ahora
13     private MyChromosome bestFitnessChromosome;
14     //Variables auxiliares para la recombinación
15     private Double[] fitnessRoulette;
16     private int[] fitnessRouletteRanking;
17
18     //Inicializa los valores
19     public MyPopulation(int size) {
20       Size = size;
21       CurrentPopulation = new MyChromosome[Size];
22       fitnessRoulette = new Double[Size];
23       fitnessRouletteRanking = new int[Size];
24       bestFitnessChromosome = null;
25     }
26
27     //Getters and setters
28     public int Size {
29       get {
30         return size;
31       }
32
33       set {
34         size = value;
35       }
36     }
37
38     public MyChromosome[] CurrentPopulation {
```

```
39        get {
40          return currentPopulation;
41        }
42
43        set {
44          currentPopulation = value;
45        }
46      }
47
48      internal MyChromosome BestFitnessChromosome {
49        get {
50          return bestFitnessChromosome;
51        }
52
53        set {
54          bestFitnessChromosome = value;
55        }
56      }
57
58      public MyChromosome getChromosomeAt(int pos) {
59        return CurrentPopulation[pos];
60      }
61
62      //Crea los individuos de la población
63      public void init() {
64        for (int aux = 0; aux < Size; aux++) {
65          MyChromosome cAux = new MyChromosome();
66          cAux.calcFitness();
67          CurrentPopulation[aux] = cAux;
68        }
69      }
70
71      /*
72      * Ordena todos los individuos de mayor a menor fitness.
73      * Una vez ordenados, el primero, si tiene menor fitness que el mejor
            individuo encontrado, se convierte en el mejor
74      * Dado que utilizo un método probabilistico para saber el fitness de una
            imagen recalculo cada vez el fitness del mejor individuo
75      * Si realmente es bueno, se mantendrá estable, si por el contrario fue
            suerte, será descartado
76      */
77      public void sort() {
78        if (BestFitnessChromosome != null) {
79          BestFitnessChromosome.calcFitness();
80        }
81
82        for (int i = 0; i < Size - 1; i++) {
83          for (int j = i + 1; j > 0; j--) {
84            if (CurrentPopulation[j - 1].Fitness < CurrentPopulation[j].Fitness ||
                  CurrentPopulation[j - 1].Fitness == Double.NaN) {
85              MyChromosome cAux = CurrentPopulation[j];
86              CurrentPopulation[j] = CurrentPopulation[j - 1];
87              CurrentPopulation[j - 1] = cAux;
88            }
89          }
90        }
91        if (BestFitnessChromosome == null) {
```

```
92          BestFitnessChromosome = CurrentPopulation[0].Clone();
93        }else if (BestFitnessChromosome.Fitness == Double.NaN) {
94          BestFitnessChromosome = CurrentPopulation[0].Clone();
95        } else if (CurrentPopulation[0].Fitness > BestFitnessChromosome.Fitness) {
96          BestFitnessChromosome = CurrentPopulation[0].Clone();
97        }
98      }
99      //Según cual se usa, incializa un método de selección u otro
100     public void prepareRoulette() {
101       if (Helper.useFitnessRoulette) {
102         this.prepareRouletteFitness();
103       } else {
104         this.prepareRouletteRanking();
105       }
106     }
107     //Según cual se usa, selecciona individuos de un método u otro
108     public int select() {
109       if (Helper.useFitnessRoulette) {
110         return selectRouletteFitness();
111       } else {
112         return selectRouletteRanking();
113       }
114
115     }
116     //Realiza el sumatario de fitness y asigna a cada individuo, en otro array,
          lo acumulado (Leer memoria para enteder como funciona)
117     public void prepareRouletteFitness() {
118       double sumatory = 0;
119       for (int i = 0; i < Size; i++) {
120         sumatory += CurrentPopulation[i].Fitness;
121         fitnessRoulette[i] = sumatory;
122       }
123     }
124     //Lanza un número al azar, recorre todo el array de fitness acumulado y,
          cuando encuentra un individuo que la suma es mayor lo selecciona
125     public int selectRouletteFitness() {
126       double aux = Helper.rnd.NextDouble() * fitnessRoulette[Size - 1];
127       for (int i = 0; i < Size; i++) {
128         if (aux < fitnessRoulette[i]) {
129           return i;
130         }
131       }
132       return 0;
133     }
134     //Igual que "prepareRouletteFitness", pero el valor acumulado es la posición
          que ocupa en el aray empezando por el final
135     public void prepareRouletteRanking() {
136       int sumatory = Size;
137       for (int i = 0; i < Size; i++) {
138         if (currentPopulation[i].Fitness == 0) {
139           fitnessRouletteRanking[i] = 0;
140         }
141         sumatory = sumatory - i;
142         fitnessRouletteRanking[i] = sumatory;
143         sumatory = sumatory + Size;
144       }
145     }
```

```
146     //Exactamente igual que "selectRouletteFitness" con distintos valores
147     public int selectRouletteRanking() {
148       int aux = Helper.rnd.Next(0, fitnessRouletteRanking[Size - 1]);
149       for (int i = 0; i < Size; i++) {
150         if (aux < fitnessRouletteRanking[i]) {
151           return i;
152         }
153       }
154       return 0;
155     }
156
157     /*
158     * Primero selecionamos los X mejores (primer for)
159     * Los siguientes, utilizan el metodo de selección y crossover (segundo for)
160     * Los ultimos, son generados al azar (tercer for)
161     * Finalmente reemplaza la población
162     */
163     public void crossover() {
164       MyChromosome[] newPopulation = new MyChromosome[Size];
165       for (int i = 0; i < Helper.elitismRange; i++) {
166         newPopulation[i] = currentPopulation[i].Clone();
167       }
168       this.prepareRoulette();
169       for (int i = Helper.elitismRange; i < Size; i = i + 2) {
170         int m = select();
171         int f = select();
172         if (m == f) {
173           f = select();
174         }
175         MyChromosome c1 = currentPopulation[m].Clone();
176         MyChromosome c2 = currentPopulation[f].Clone();
177         if (Helper.rnd.NextDouble() < Helper.crossoverRate) {
178           c1.valueSwapCrossover(c2);
179         }
180
181         newPopulation[i] = c1;
182         newPopulation[i + 1] = c2;
183       }
184
185       currentPopulation = newPopulation;
186     }
187     //Por individuo lanza un numero entre 0 y 1, si es menor que el ratio de
          mutación, muta el individuo
188     public void mutation() {
189       for (int i = Helper.elitismRange; i < Size; i++) {
190         if (Helper.rnd.NextDouble() < Helper.mutationRate) {
191           currentPopulation[i].valueMutaton();
192         }
193         currentPopulation[i].calcFitness();
194       }
195     }
196   }
197 }
```

A.4 MainWindow.xaml

En C♯ existe una librería llamada Windows Presentation Foundation (WPF) muy completa y fácil de usar para la creación de interfaces gráfica de usuario. Esta, usa un archivo con estructura eXtensible Markup Language (XML) para definir los elementos de la Grafical User Interface (GUI), como botones, etiquetas, cuadros de texto, e incluso la propia ventana.

```
1  <Window x:Class="panoramic.MainWindow"
2      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6      xmlns:local="clr-namespace:panoramic"
7      mc:Ignorable="d"
8      Title="MainWindow" Height="850" Width="1300" Background="Gray">
9    <Grid>
10     <Grid.ColumnDefinitions>
11       <ColumnDefinition Width="*"/>
12       <ColumnDefinition Width="125"/>
13     </Grid.ColumnDefinitions>
14     <Grid.RowDefinitions>
15       <RowDefinition Height="*" />
16       <RowDefinition Height="100" />
17     </Grid.RowDefinitions>
18
19     <Button x:Name="runEpochButton" Content="Run Epoch"
           HorizontalAlignment="Center"  VerticalAlignment="Center"
           Margin="7.5,-30,0,0" Width="75" Grid.Row="1" Grid.Column="1"
           Click="runEpoch"/>
20     <Button x:Name="runButton" Content="Run" HorizontalAlignment="Center"
           VerticalAlignment="Center" Margin="7.5,30,0,0" Width="75" Grid.Row="1"
           Grid.Column="1" Click="run"/>
21     <ScrollViewer Grid.Column="0" HorizontalScrollBarVisibility="Auto"
           VerticalScrollBarVisibility="Auto">
22     <Grid>
23     <Image x:Name="originalImage" HorizontalAlignment="Left"
           VerticalAlignment="Top" Stretch="None" Grid.Row="0" Grid.Column="0"/>
24     <Image x:Name="originalImage2" HorizontalAlignment="Left"
           VerticalAlignment="Top" Stretch="None" Grid.Row="0" Grid.Column="0"/>
25     </Grid>
26     </ScrollViewer>
27     <ProgressBar x:Name="progressBar" HorizontalAlignment="Left" Height="17"
           Margin="10,35,0,0" Grid.Row="1" VerticalAlignment="Top" Width="1149"/>
28     <Label x:Name="generationInfo" HorizontalContentAlignment="Right" Content=""
           HorizontalAlignment="Left" Margin="1007,52,0,0" VerticalAlignment="Top"
           Grid.Row="1" Width="152"/>
29     <Label x:Name="fitnessInfo" Content="" HorizontalAlignment="Left"
           Margin="8,52,0,0" VerticalAlignment="Top" Grid.Row="1" Width="277"/>
30
31   </Grid>
32 </Window>
```

A.5 MainWindow.xaml.cs

Establece las acciones y eventos que existen en la interfaz gráfica de WPF. En esta clase es donde se inicializa el algoritmo genético, como un proceso que se ejecuta en segundo plano, para que no interfiera y bloquea la GUI.

```
1  using System;
2  using System.Windows;
3  using System.ComponentModel;
4
5  namespace panoramic {
6
7    /// <summary>
8    /// Interaction logic for MainWindow.xaml
9    /// </summary>
10   public partial class MainWindow : Window {
11
12     MyPopulation population;
13     BackgroundWorker bw = new BackgroundWorker();
14
15     public MainWindow() {
16       InitializeComponent();
17       originalImage.Source = Helper.bitmapImage;
18       originalImage.Width = Helper.bitmapImage.Width;
19       originalImage.Height = Helper.bitmapImage.Height;
20
21       //Inicializa los parametros necesarios para poder ejecutar el proceso en
             "background"
22       bw.DoWork += bw_DoWork;
23       bw.ProgressChanged += bw_ProgressChanged;
24       bw.RunWorkerCompleted += bw_RunWorkerCompleted;
25       bw.WorkerReportsProgress = true;
26     }
27
28     //Hace una sola epoca/iteracion. Se puede poner otra cantidad para probar
29     public void runEpoch(object sender, RoutedEventArgs e) {
30       int i = 0;
31       do {
32         population.sort();
33         population.crossover();
34         population.mutation();
35         i++;
36       } while (i<1);
37       population.sort();
38       originalImage2.Source = Helper.bitmapImage2;
39       //Wpf cambia la resolucion de la imagen, asi que hay que hacer un pequeño
             cambio para que el offset se corresponda
40       double dpiDiffX = Helper.bitmapImage2.Width /
             Helper.bitmapImage2.PixelWidth;
41       double dpiDiffY = Helper.bitmapImage2.Height /
             Helper.bitmapImage2.PixelHeight;
42       originalImage2.Margin = new
             Thickness(population.BestFitnessChromosome.XValue * dpiDiffX,
             population.BestFitnessChromosome.YValue * dpiDiffY, 0, 0);
43     }
44     //Ejecuta el algoritmo entero
45     public void run(object sender, RoutedEventArgs e) {
```

```
46        runButton.IsEnabled = false;
47        population = new MyPopulation(Helper.populationSize);
48
49        population.init();
50        bw.RunWorkerAsync();
51      }
52
53      //Ejecuta el algoritmo en "background"
54      private void bw_DoWork(object sender, DoWorkEventArgs e) {
55        int i = 0;
56        population.sort();
57        do {
58          population.crossover();
59          population.mutation();
60          population.sort();
61          i++;
62          //Cunado se ha llegado a la mitad de las iteraciones, se intenta ajustar
                la imagen reduciendo la mutacion
63          if (i == Helper.numberOfGenerations / 2) {
64            Helper.mutationRange = 50;
65          }
66          bw.ReportProgress(i);
67          double a = population.BestFitnessChromosome.Fitness /
                Helper.pointsToCompare;
68        } while (i < Helper.numberOfGenerations &&
              population.BestFitnessChromosome.Fitness / Helper.pointsToCompare <
              Helper.threshold);
69      }
70      //Actualiza la información de porcentage
71      private void bw_ProgressChanged(object sender, ProgressChangedEventArgs e) {
72        double percent = Helper.numberOfGenerations / 100;
73        generationInfo.Content = e.ProgressPercentage + "/" +
              Helper.numberOfGenerations;
74        progressBar.Value = Convert.ToInt32(e.ProgressPercentage / percent);
75        fitnessInfo.Content = population.BestFitnessChromosome.Fitness /
              Helper.pointsToCompare;
76      }
77      //Ha acabado, muestra la solución
78      private void bw_RunWorkerCompleted(object sender,
            System.ComponentModel.RunWorkerCompletedEventArgs e) {
79        population.sort();
80        originalImage2.Source = Helper.bitmapImage2;
81        //Wpf cambia la resolucion de la imagen, asi que hay que hacer un pequeño
              cambio para que el offset se corresponda
82        double dpiDiffX = Helper.bitmapImage2.Width /
              Helper.bitmapImage2.PixelWidth;
83        double dpiDiffY = Helper.bitmapImage2.Height /
              Helper.bitmapImage2.PixelHeight;
84        originalImage2.Margin = new
              Thickness(population.BestFitnessChromosome.XValue * dpiDiffX,
              population.BestFitnessChromosome.YValue * dpiDiffY, 0, 0);
85        runButton.IsEnabled = true;
86
87        progressBar.Value = 100;
88      }
89    }
90  }
```

APÉNDICE B

CÓDIGO ALGORITMOS GENÉTICOS EN CRIPTOGRAFÍA

B.1 Helper.cs

Clase que tiene todas las variables que el usuario puede modificar para alterar el comportamiento del algoritmo. Están separadas para facilitar la localización de las mismas.

```
1
2  using System;
3  using System.IO;
4  using System.Text;
5  using System.Text.RegularExpressions;
6
7  namespace crypto {
8    class Helper {
9
10     //Variables de propias del GA
11
12     //Tamaño de la población
13     public static int populationSize = 64;
14     //Probabilidad de que dos individuos se reproduzcan (rango 0-1)
15     public static double crossoverRate = 0.5;
16     //Probabilidad de mutación de cada individuo (rango 0-1)
17     public static double mutationRate = 1;
18     //Número de individuos elitistas
19     public static int elitismRange = 8;
20     //Indica si se usa ruleta por fitness o ranking (true/false, respectivamente)
21     public static bool useFitnessRoulette = false;
22     //Nuevo individuos que se crean al azar en cada generación
23     public static int newIndividuals = 12;
24     //Fitness aceptado como solución
25     public static double fitnessThreshold = 0.25;
26     //Número de generaciones que se hará si no se encuentra el fitness antes
27     public static int numberOfGenerations = 500;
28
```

```
29
30      //Clase de C# para obtener nuemero pseudoaleatorios
31      public static Random rnd = new Random();
32      //Texto a desencriptar, si es en castellano hay que quitar acentos
33      public static String originalText = "En␣efecto,␣rematado␣ya␣su␣juicio,␣vino␣
            a␣dar␣en␣el␣más␣extraño␣pensamiento␣que␣jamás␣dio␣loco␣en␣el␣mundo,␣y␣
            fue␣que␣le␣pareció␣convenible␣y␣necesario,␣así␣para␣el␣aumento␣de␣su␣
            honra,␣como␣para␣el␣servicio␣de␣su␣república,␣hacerse␣caballero␣andante,␣
            e␣irse␣por␣todo␣el␣mundo␣con␣sus␣armas␣y␣caballo␣a␣buscar␣las␣aventuras,␣
            y␣a␣ejercitarse␣en␣todo␣aquello␣que␣él␣había␣leído,␣que␣los␣caballeros␣
            andantes␣se␣ejercitaban,␣deshaciendo␣todo␣género␣de␣agravio,␣y␣
            poniéndose␣en␣ocasiones␣y␣peligros,␣donde␣acabándolos,␣cobrase␣eterno␣
            nombre␣y␣fama.";
34      //Texto encriptado al azar
35      public static String cryptedText = "";
36      //Clave de encriptado original
37      public static char[] key = new Char[26];
38      //Contienen las frecuencias de grupos de dos y tres letras seguidas
39      public static FrequencyDouble fd = new FrequencyDouble();
40      public static FrequencyTriple ft = new FrequencyTriple();
41      //Texto con el cual se inician las tablas de frecuencias
42      public static string quijote = "";
43
44      //Encripta el texto original y lee de un fichero un texto para inciar las
            tablas de frecuencias de la lengua
45      public static void initStaticComponents() {
46        cryptText();
47        quijote = File.ReadAllText("../../textoPrueba.txt",
              Encoding.GetEncoding("iso-8859-1"));
48        quijote = Helper.RemoveDiacritics(quijote);
49        quijote = Regex.Replace(quijote, "[0-9]", "");
50        fd.initCharFrequencyDoubleTable(quijote);
51        ft.initCharFrequencyTripleTable(quijote);
52      }
53      //Transforma un caracter en el indice de las tablas (cualquier simbolo de
            puntuación es tomado con espacio)
54      public static int getIndex(char c) {
55        byte asciiBytes = Encoding.ASCII.GetBytes(Char.ToLower(c) + "")[0];
56        if (asciiBytes == 63) {
57          return /*26*/13;
58        } else if (asciiBytes == 32) {
59          return 26;
60        }
61        int index = (int)asciiBytes - 97;
62        if ((index < 0 || index > 25)) {
63          return 26;
64        }
65        return index;
66      }
67      //Crea las llaves de ecnriptado. Array en orden alfabetico y luego lo
            desordena, es mas facil asi
68      public static void cryptText() {
69        for (int i = 0; i < key.Length; i++) {
70          key[i] = (char)(97 + i);
71        }
72        for (int i = 0; i < key.Length - 1; i++) {
73          int j = Helper.rnd.Next(i + 1, key.Length);
```

```
74          char t = key[j];
75          key[j] = key[i];
76          key[i] = t;
77        }
78        //Encripta el texto segun la clave creada
79        originalText = RemoveDiacritics(originalText);
80        foreach (char c in originalText) {
81          char aux;
82          int index = getIndex(c);
83          if (index == -1 || index == 26) {
84            aux = c;
85          }else {
86            aux = key[index];
87          }
88
89          cryptedText += aux;
90        }
91      }
92      //Elimina acentos del texto para facilitar las tablas
93      public static string RemoveDiacritics(string value) {
94        if (!string.IsNullOrWhiteSpace(value)) {
95          string stFormD = value.Normalize(NormalizationForm.FormD);
96          int len = stFormD.Length;
97          StringBuilder sb = new StringBuilder();
98
99          for (int i = 0; i < len; i++) {
100           System.Globalization.UnicodeCategory uc =
                  System.Globalization.CharUnicodeInfo.GetUnicodeCategory(stFormD[i]);
101           if (uc != System.Globalization.UnicodeCategory.NonSpacingMark) {
102             sb.Append(stFormD[i]);
103           }
104         }
105
106         return (sb.ToString().Normalize(NormalizationForm.FormC));
107       } else {
108         return value;
109       }
110     }
111
112   }
113 }
```

B.2 MyChromosome.cs

Clase que representa los datos de los cromosomas, así como las operaciones que se realizan sobre estas. Inicializa, reproduce, muta y calcula el “fitness” de los individuos.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace crypto {
8   //Individuo en castellano
9   class MyChromosome {
```

```
10
11      //Texto desencryptado
12      private String text = "";
13      //Clave de desencryptado
14      private char[] charKey = new char[26];
15      //Fitness del individuo
16      private double fitness = -1;
17
18      //Crea un nuevo individuo. Crea la clave (primero en orden y luego
            desordena) y calcula el fitness
19      public MyChromosome() {
20        for (int i = 0; i < CharKey.Length; i++) {
21          CharKey[i] = (char)(97 + i);
22        }
23        for (int i = 0; i < CharKey.Length - 1; i++) {
24          int j = Helper.rnd.Next(i + 1, CharKey.Length);
25          char t = CharKey[j];
26          CharKey[j] = CharKey[i];
27          CharKey[i] = t;
28        }
29        calcFitness();
30      }
31      //Crea un indivuduo con una clave en concreto
32      public MyChromosome(char[] key) {
33        key.CopyTo(CharKey, 0);
34        decrypt();
35        calcFitness();
36      }
37      //Gets y sets de las propiedades
38      public double Fitness {
39        get {
40          if (fitness == -1) {
41            calcFitness();
42          }
43          return fitness;
44        }
45
46        set {
47          fitness = value;
48        }
49      }
50
51      public char[] CharKey {
52        get {
53          return charKey;
54        }
55
56        set {
57          charKey = value;
58        }
59      }
60
61      public string Text {
62        get {
63          return text;
64        }
65
```

```
66        set {
67          text = value;
68        }
69      }
70
71      public char get(int i) {
72        return CharKey[i];
73      }
74
75      public void set(int i, char c) {
76        CharKey[i] = c;
77      }
78
79      //Desencrypta el texto según la clave de desencriptado de este individuo
80      public void decrypt() {
81        Text = "";
82        foreach (char c in Helper.cryptedText) {
83          char aux;
84          int index = Helper.getIndex(c);
85          if (index == -1 || index == 26) {
86            aux = c;
87          } else {
88            aux = CharKey[index];
89          }
90          Text += aux;
91        }
92      }
93
94      //Calcula el fitness del texto (desencrpta primero)
95      public void calcFitness() {
96        double total = 0;
97        decrypt();
98        for (int i = 0; i < Text.Length; i++) {
99          //Tiene en cuenta todas las convinaciones de dos y de tres y sus
                frecuencias
100         if (i > 0) {
101           double d = Helper.fd.getFrequency(Text[i - 1], Text[i]);
102           if (d > 0) {
103             total += 0.4 * d;
104           } else {
105             total -= 0.05;
106           }
107         }else {
108           double d = Helper.fd.getFrequency(' ', Text[i]);//Simula comienzo de
                palabra
109           if (d > 0) {
110             total += 0.4 * d;
111           } else {
112             total -= 0.05;
113           }
114         }
115         if (i > 1) {
116           double d = Helper.ft.getFrequency(Text[i - 2], Text[i - 1], Text[i]);
117           if (d > 0) {
118             total += 0.6 * d;
119           } else {
120             total -= 0.03;
```

```
121           }
122         }
123       }
124       fitness = total;
125     }
126
127
128     /*
129     * Realiza la recombinación con otro individuo.
130     * Debido a que no se pueden repetir las letras en la clave, es mucho mas
            dificil de explicar el algoritmo (no s epuede usar ninguno de los
            básicos)
131     * Se explica en la memoria ya que es demasiado extenso
132     */
133     public void valueSwapCrossover(MyChromosome p2, out MyChromosome c1, out
            MyChromosome c2) {
134       c2 = this.Clone();
135       c1 = p2.Clone();
136
137
138       int s1 = Helper.rnd.Next(0, CharKey.Length);
139       int s2 = s1;
140       int temp = 0;
141
142       c1.set(s1, this.get(s1));
143       while (p2.get(s2) != this.get(s1)) {
144         temp = this.search(p2.get(s2));
145         c1.set(temp, p2.get(s2));
146         s2 = temp;
147       }
148
149       s2 = s1;
150       c2.set(s1, p2.get(s1));
151       while (this.get(s2) != p2.get(s1)) {
152         temp = p2.search(this.get(s2));
153         c2.set(temp, this.get(s2));
154         s2 = temp;
155       }
156
157     }
158     //Devuelva la posición del caracter en la clave
159     public int search(char c) {
160       for (int i = 0; i < CharKey.Length ; i++) {
161         if (this.get(i) == c) return i;
162       }
163       return -1;
164     }
165     //Muta e individuo. Cambia una letra por otra el la clave (se puede repetir
            mas de una vez)
166     public void valueMutaton() {
167       int s1;
168       int s2;
169       char aux;
170       int times = Helper.rnd.Next(1, 5);
171       for (int i = 0; i < times; i++) {
172         s1 = Helper.rnd.Next(0, CharKey.Length / 2);
173         s2 = Helper.rnd.Next(CharKey.Length / 2, CharKey.Length);
```

```
174
175
176
177           aux = CharKey[s1];
178           CharKey[s1] = CharKey[s2];
179           CharKey[s2] = aux;
180         }
181       }
182       //Clona el individuo
183       public MyChromosome Clone() {
184         return new MyChromosome(this.CharKey);
185       }
186     }
187 }
```

B.3 MyPopulation.cs

Clase que representa la población completa que se esta estudiando, así como las operaciones necesarios para que esta evolucione. Inicializa y establece como se reproducen los individuos, así como cuales de ellos se reproducen, pasan a la siguiente población sin ningún cambio, desaperecen, mutan...

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace crypto {
8    class MyPopulation {
9      //Población de todos los individuos
10     private MyChromosome[] currentPopulation;
11     private int size;
12     //El mejor indivuduo encontrado hasta ahora
13     private MyChromosome bestFitnessChromosome;
14     //Variables auxiliares para la recombinación
15     private Double[] fitnessRoulette;
16     private int[] fitnessRouletteRanking;
17
18     //Inicializa los valores
19     public MyPopulation(int size) {
20       Size = size;
21       CurrentPopulation = new MyChromosome[Size];
22       fitnessRoulette = new Double[Size];
23       fitnessRouletteRanking = new int[Size];
24     }
25
26     //Getters and setters
27     public int Size {
28       get {
29         return size;
30       }
31
32       set {
33         size = value;
```

```
34        }
35      }
36
37      public MyChromosome[] CurrentPopulation {
38        get {
39          return currentPopulation;
40        }
41
42        set {
43          currentPopulation = value;
44        }
45      }
46
47      internal MyChromosome BestFitnessChromosome {
48        get {
49          return bestFitnessChromosome;
50        }
51
52        set {
53          bestFitnessChromosome = value;
54        }
55      }
56
57      public MyChromosome getChromosomeAt(int pos) {
58        return CurrentPopulation[pos];
59      }
60
61      //Crea los individuos de la población
62      public void init() {
63        for (int aux = 0; aux < Size; aux++) {
64          MyChromosome cAux = new MyChromosome();
65          cAux.calcFitness();
66          CurrentPopulation[aux] = cAux;
67        }
68      }
69
70      /*
71      * Ordena todos los individuos de mayor a menor fitness.
72      * Una vez ordenados, el primero, si tiene menor fitness que el mejor
            individuo encontrado, se convierte en el mejor
73      */
74      public void sort() {
75        for (int i = 0; i < Size - 1; i++) {
76          for (int j = i + 1; j > 0; j--) {
77            if (CurrentPopulation[j - 1].Fitness < CurrentPopulation[j].Fitness ||
                  CurrentPopulation[j - 1].Fitness == Double.NaN) {
78              MyChromosome cAux = CurrentPopulation[j];
79              CurrentPopulation[j] = CurrentPopulation[j - 1];
80              CurrentPopulation[j - 1] = cAux;
81            }
82          }
83        }
84        if (BestFitnessChromosome == null) {
85          BestFitnessChromosome = CurrentPopulation[0].Clone();
86        }else if (BestFitnessChromosome.Fitness == Double.NaN) {
87          BestFitnessChromosome = CurrentPopulation[0].Clone();
88        } else if (CurrentPopulation[0].Fitness > BestFitnessChromosome.Fitness) {
```

```
89          BestFitnessChromosome = CurrentPopulation[0].Clone();
90        }
91      }
92      //Según cual se usa, incializa un método de selección u otro
93      public void prepareRoulette() {
94        if (Helper.useFitnessRoulette) {
95          this.prepareRouletteFitness();
96        } else {
97          this.prepareRouletteRanking();
98        }
99      }
100     //Según cual se usa, selecciona individuos de un método u otro
101     public int select() {
102       if (Helper.useFitnessRoulette) {
103         return selectRouletteFitness();
104       } else {
105         return selectRouletteRanking();
106       }
107
108     }
109     //Realiza el sumatario de fitness y asigna a cada individuo, en otro array,
            lo acumulado (Leer memoria para enteder como funciona)
110     public void prepareRouletteFitness() {
111       double sumatory = 0;
112       for (int i = 0; i < Size; i++) {
113         sumatory += CurrentPopulation[i].Fitness;
114         fitnessRoulette[i] = sumatory;
115       }
116     }
117     //Lanza un número al azar, recorre todo el array de fitness acumulado y,
            cuando encuentra un individuo que la suma es mayor lo selecciona
118     public int selectRouletteFitness() {
119       double aux = Helper.rnd.NextDouble() * fitnessRoulette[Size - 1];
120       for (int i = 0; i < Size; i++) {
121         if (aux < fitnessRoulette[i]) {
122           return i;
123         }
124       }
125       return 0;
126     }
127     //Igual que "prepareRouletteFitness", pero el valor acumulado es la posición
            que ocupa en el aray empezando por el final
128     public void prepareRouletteRanking() {
129       int sumatory = Size;
130       for (int i = 0; i < Size; i++) {
131         sumatory = sumatory - i;
132         fitnessRouletteRanking[i] = sumatory;
133         sumatory = sumatory + Size;
134       }
135     }
136     //Exactamente igual que "selectRouletteFitness" con distintos valores
137     public int selectRouletteRanking() {
138       int aux = Helper.rnd.Next(0, fitnessRouletteRanking[Size - 1]);
139       for (int i = 0; i < Size; i++) {
140         if (aux < fitnessRouletteRanking[i]) {
141           return i;
142         }
```

```
143       }
144       return 0;
145     }
146
147     /*
148     * Primero selecionamos los X mejores (primer for)
149     * Los siguientes, utilizan el metodo de selección y crossover (segundo for)
150     * Los ultimos, son generados al azar (tercer for)
151     * Finalmente reemplaza la población
152     */
153     public void crossover() {
154       MyChromosome[] newPopulation = new MyChromosome[Size];
155       for (int i = 0; i < Helper.elitismRange; i++) {
156         newPopulation[i] = currentPopulation[i].Clone();
157       }
158       this.prepareRoulette();
159       for (int i = Helper.elitismRange; i < Size - Helper.newIndividuals; i = i
              + 2) {
160         int m = select();
161         int f = select();
162         if (m == f) {
163           f = select();
164         }
165         MyChromosome p1 = currentPopulation[m].Clone();
166         MyChromosome p2 = currentPopulation[f].Clone();
167         MyChromosome c1 = null;
168         MyChromosome c2 = null;
169
170         if (Helper.rnd.NextDouble() < Helper.crossoverRate) {
171           p1.valueSwapCrossover(p2, out c1, out c2);
172         }else {
173           c1 = p1;
174           c2 = p2;
175         }
176
177         newPopulation[i] = c1;
178         newPopulation[i + 1] = c2;
179       }
180       for (int i = Size - Helper.newIndividuals; i < Size; i++) {
181         newPopulation[i] = new MyChromosome();
182       }
183
184       currentPopulation = newPopulation;
185     }
186
187     //Por individuo lanza un numero entre 0 y 1, si es menor que el ratio de
            mutación, muta el individuo
188     public void mutation() {
189       for (int i = Helper.elitismRange; i < Size; i++) {
190         if (Helper.rnd.NextDouble() < Helper.mutationRate) {
191           currentPopulation[i].valueMutaton();
192         }
193         currentPopulation[i].calcFitness();
194       }
195     }
196   }
197 }
```

B.4 MainWindow.xaml

En C♯ existe una librería llamada WPF muy completa y fácil de usar para la creación de interfaces gráfica de usuario. Esta, usa un archivo con estructura XML para definir los elementos de la GUI, como botones, etiquetas, cuadros de texto, e incluso la propia ventana.

```
1  <Window x:Class="crypto.MainWindow"
2    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6    xmlns:local="clr-namespace:crypto"
7    mc:Ignorable="d"
8    Title="MainWindow" Height="850" Width="1300">
9    <Grid>
10     <Grid.ColumnDefinitions>
11       <ColumnDefinition Width="*"/>
12       <ColumnDefinition Width="125"/>
13     </Grid.ColumnDefinitions>
14     <Grid.RowDefinitions>
15       <RowDefinition Height="*" />
16       <RowDefinition Height="100" />
17     </Grid.RowDefinitions>
18     <Button x:Name="runButton" Content="Run" HorizontalAlignment="Center"
           VerticalAlignment="Center" Margin="7.5,30,0,0" Width="75" Grid.Row="1"
           Grid.Column="1" Click="run"/>
19     <TextBox x:Name="textBoxOriginal" HorizontalAlignment="Left" Height="450"
           Margin="10,209,0,0" TextWrapping="Wrap" Text="TextBox"
           VerticalAlignment="Top" Width="350"/>
20     <TextBox x:Name="textBoxCrypted" HorizontalAlignment="Left" Height="450"
           Margin="390,209,0,0" TextWrapping="Wrap" Text="TextBox"
           VerticalAlignment="Top" Width="350"/>
21     <TextBox x:Name="textBoxDecrypted" HorizontalAlignment="Left" Height="450"
           Margin="780,209,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top"
           Width="350"/>
22     <Label x:Name="labelOriginal" Content="Texto original"
           HorizontalAlignment="Left" Margin="10,178,0,0" VerticalAlignment="Top"
           RenderTransformOrigin="0.167,-0.242"/>
23     <Label x:Name="labelCrypted" Content="Texto encriptado"
           HorizontalAlignment="Left" Margin="390,178,0,0" VerticalAlignment="Top"
           RenderTransformOrigin="0.167,-0.242"/>
24     <Label x:Name="labelDecrypted" Content="Texto desencriptado"
           HorizontalAlignment="Left" Margin="780,178,0,0" VerticalAlignment="Top"
           RenderTransformOrigin="0.167,-0.242"/>
25
26     <Label x:Name="labelCryptedKeyText" Content="Clave de encriptado: "
           HorizontalAlignment="Left" Margin="10,684,0,0" VerticalAlignment="Top"/>
27     <Label x:Name="labelCryptedKey" Content="" HorizontalAlignment="Left"
           Margin="137,684,0,0" VerticalAlignment="Top" Width="436"/>
28     <Label x:Name="labelDecryptedKeyText" Content="Clave de desencriptado: "
           HorizontalAlignment="Left" Margin="587,684,0,0" VerticalAlignment="Top"/>
29     <Label x:Name="labelDecryptedKey" Content="" HorizontalAlignment="Left"
           Margin="731,684,0,0" VerticalAlignment="Top" Width="399"/>
30
31     <Label x:Name="populationLabel" Content="Tamaño de población"
           HorizontalAlignment="Left" Margin="10,10,0,0" VerticalAlignment="Top"/>
```

```
32      <Label x:Name="numberOfGenerationsLabel" Content="Número de generaciones"
            HorizontalAlignment="Left" Margin="8,41,0,0" VerticalAlignment="Top"/>
33      <Label x:Name="fitnessThresholdLabel" Content="Umbral de fitness"
            HorizontalAlignment="Left" Margin="8,72,0,0" VerticalAlignment="Top"/>
34      <Label x:Name="crossoverRateLabel" Content="Ratio de recombinación"
            HorizontalAlignment="Left" Margin="390,37,0,0" VerticalAlignment="Top"/>
35      <Label x:Name="mutationRateLabel" Content="Probabilidad de mutación"
            HorizontalAlignment="Left" Margin="390,10,0,0" VerticalAlignment="Top"/>
36      <Label x:Name="elitismRateLabel" Content="Número de elitistas"
            HorizontalAlignment="Left" Margin="390,68,0,0" VerticalAlignment="Top"/>
37
38      <TextBox x:Name="populationText" Text="64" HorizontalAlignment="Left"
            Margin="139,10,0,0" VerticalAlignment="Top" Width="221"/>
39      <TextBox x:Name="numberOfGenerationsText" Text="1000"
            HorizontalAlignment="Left" Margin="157,41,0,0" VerticalAlignment="Top"
            Width="203"/>
40      <TextBox x:Name="fitnessThresholdText" Text="0,245"
            HorizontalAlignment="Left" Margin="115,72,0,0" VerticalAlignment="Top"
            Width="245"/>
41      <TextBox x:Name="mutationRateText" Text="1" HorizontalAlignment="Left"
            Margin="538,10,0,0" VerticalAlignment="Top" Width="202"/>
42      <TextBox x:Name="crossoverRateText" Text="0,5" HorizontalAlignment="Left"
            Margin="526,41,0,0" VerticalAlignment="Top" Width="214"/>
43      <TextBox x:Name="elitismRateText" Text="12" HorizontalAlignment="Left"
            Margin="509,72,0,0" VerticalAlignment="Top" Width="231"/>
44
45      <Label x:Name="newIndividualsLabel" Content="Nuevo de individuos creados"
            HorizontalAlignment="Left" Margin="780,23,0,0" VerticalAlignment="Top"/>
46      <TextBox x:Name="newIndividualsText" Text="14" HorizontalAlignment="Left"
            Margin="943,23,0,0" VerticalAlignment="Top" Width="187"/>
47      <ProgressBar x:Name="progressBar" HorizontalAlignment="Left" Height="17"
            Margin="10,35,0,0" Grid.Row="1" VerticalAlignment="Top" Width="1149"/>
48      <Label x:Name="fitnessInfo" Content="" HorizontalAlignment="Left"
            Margin="8,52,0,0" VerticalAlignment="Top" Grid.Row="1" Width="277"/>
49      <Label x:Name="generationInfo" HorizontalContentAlignment="Right" Content=""
            HorizontalAlignment="Left" Margin="1007,52,0,0" VerticalAlignment="Top"
            Grid.Row="1" Width="152"/>
50
51    </Grid>
52  </Window>
```

B.5 MainWindow.xaml.cs

Establece las acciones y eventos que existen en la interfaz gráfica de WPF. En esta clase es donde se inicializa el algoritmo genético, como un proceso que se ejecuta en segundo plano, para que no interfiera y bloquea la GUI.

```
1  using System;
2  using System.Windows;
3  using System.ComponentModel;
4
5  namespace crypto {
6    /// <summary>
7    /// Interaction logic for MainWindow.xaml
8    /// </summary>
```

```
9   public partial class MainWindow : Window {
10
11    MyPopulation population;
12    BackgroundWorker bw = new BackgroundWorker();
13
14    public MainWindow() {
15      InitializeComponent();
16      //Incia los datos auxilires
17      Helper.initStaticComponents();
18      textBoxOriginal.Text = Helper.originalText;
19      textBoxCrypted.Text = Helper.cryptedText;
20      String keyAux = "";
21      foreach (char c in Helper.key) {
22        keyAux += c;
23      }
24      labelCryptedKey.Content = keyAux;
25      progressBar.Maximum = 100;
26
27      //Inicializa los parametros necesarios para poder ejecutar el proceso en
            "background"
28      bw.DoWork += bw_DoWork;
29      bw.ProgressChanged += bw_ProgressChanged;
30      bw.RunWorkerCompleted += bw_RunWorkerCompleted;
31      bw.WorkerReportsProgress = true;
32    }
33
34    public void run(object sender, RoutedEventArgs e) {
35      runButton.IsEnabled = false;
36      //Coge las variables introducidas por el usuario
37      Helper.populationSize = Int32.Parse(populationText.Text);
38      Helper.crossoverRate = Double.Parse(crossoverRateText.Text);
39      Helper.mutationRate = Double.Parse(mutationRateText.Text);
40      Helper.elitismRange = Int32.Parse(elitismRateText.Text);
41      Helper.newIndividuals = Int32.Parse(newIndividualsText.Text);
42      Helper.fitnessThreshold = Double.Parse(fitnessThresholdText.Text);
43      Helper.numberOfGenerations = Int32.Parse(numberOfGenerationsText.Text);
44      //Inicia la población y ejecuta el algortimo
45      population = new MyPopulation(Helper.populationSize);
46      population.init();
47      bw.RunWorkerAsync();
48
49    }
50
51    //Ejecuta el algoritmo en "background"
52    private void bw_DoWork(object sender, DoWorkEventArgs e) {
53      double max = (Helper.cryptedText.Length - 1) * Helper.fd.max * 0.4 +
            (Helper.cryptedText.Length - 2) * Helper.ft.max * 0.6;
54      int i = 0;
55
56      do {
57        population.sort();
58        population.crossover();
59        population.mutation();
60        i++;
61        bw.ReportProgress(i);
62      } while (i < Helper.numberOfGenerations &&
            population.BestFitnessChromosome.Fitness / max <
```

```
            Helper.fitnessThreshold);
63      }
64      //Actualiza la información de porcentage
65      private void bw_ProgressChanged(object sender, ProgressChangedEventArgs e) {
66        double max = (Helper.cryptedText.Length - 1) * Helper.fd.max * 0.4 +
              (Helper.cryptedText.Length - 2) * Helper.ft.max * 0.6;
67        double percent = Helper.numberOfGenerations / 100;
68        generationInfo.Content =
              e.ProgressPercentage+"/"+Helper.numberOfGenerations;
69        progressBar.Value = Convert.ToInt32(e.ProgressPercentage / percent);
70        fitnessInfo.Content = population.BestFitnessChromosome.Fitness / max;
71      }
72      //Ha acabado, muestra la solución
73      private void bw_RunWorkerCompleted(object sender,
            System.ComponentModel.RunWorkerCompletedEventArgs e) {
74        runButton.IsEnabled = true;
75        population.sort();
76        progressBar.Value = 100;
77        String keyAux = "";
78        foreach (char c in population.BestFitnessChromosome.CharKey) {
79          keyAux += c;
80        }
81        labelDecryptedKey.Content = keyAux;
82        textBoxDecrypted.Text = population.BestFitnessChromosome.Text;
83      }
84    }
85  }
```

B.6 Tablas de frecuencias

Son dos clases que permiten leer el texto y analizar las probabilidades de conjuntos de dos y tres letras en la lengua castellana. Se usan para establecer un valor de “fitness” de los cromosomas.

B.6.1 FrequencyDouble.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace crypto {
8    class FrequencyDouble {
9      /*
10     * Esta clase contiene las frecuencias de combinacion de dos letras de la
           lengua que se esta encriptando y desencriptando
11     * Los metodos son bastante fáciles de entender, cada vez que una combinación
           es leída del texto se incrementa
12     * El valor en la tabla "charFrequencyDouble", cuando el texto ha acabado se
           divide por el total para tenerlo en porcentaje
13     */
14     public double[,] charFrequencyDouble = new double[27, 27];
15     public double max = 0;
```

```
16
17      public FrequencyDouble() {
18
19      }
20      //Dadas dos letras devuelve su valor
21      public double getFrequency(char p, char c) {
22        int current = Helper.getIndex(c);
23        int previous = Helper.getIndex(p);
24        if (previous == -1 || current == -1) {
25          return 0;
26        } else if (previous == 26 && current == 26) {
27          return 0;
28        }
29        return charFrequencyDouble[previous, current];
30      }
31      //Incializa la tabla con el texto dado
32      public void initCharFrequencyDoubleTable(String s) {
33        int total = 0;
34        for (int i = 0; i < s.Length; i++) {
35          if (i > 0) {
36
37            int current = Helper.getIndex(s[i]);
38            int previous = Helper.getIndex(s[i - 1]);
39            if (previous != -1 && current != -1) {
40              if (previous != 26 || current != 26) {
41                total++;
42                charFrequencyDouble[previous, current]++;
43              }
44            }
45          }
46        }
47        for (int i = 0; i < 27; i++) {
48          for (int j = 0; j < 27; j++) {
49            charFrequencyDouble[i, j] = charFrequencyDouble[i, j] / total * 100;
50            if (charFrequencyDouble[i, j] > max) {
51              max = charFrequencyDouble[i, j];
52            }
53          }
54        }
55      }
56    }
57  }
```

B.6.2 FrequencyTriple.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace crypto {
8    class FrequencyTriple {
9      /*
10     * Esta clase contiene las frecuencias de combinacion de cuatro letras de la
           lengua que se esta encriptando y desencriptando
```

```
11      * Los metodos son bastante fáciles de entender, cada vez que una combinación
            es leída del texto se incrementa
12      * El valor en la tabla "charFrequencyDouble", cuando el texto ha acabado se
            divide por el total para tenerlo en porcentaje
13      */
14      public double[,,] charFrequencyTriple = new double[27, 27, 27];
15      public double max = 0;
16
17      public FrequencyTriple() {
18
19      }
20      //Dadas tres letras devuelve su valor
21      public double getFrequency(char pp, char p, char c) {
22        int current = Helper.getIndex(c);
23        int previous = Helper.getIndex(p);
24        int previousP = Helper.getIndex(pp);
25        if (previous == -1 || current == -1 || previousP == -1) {
26          return 0;
27        }else if (previous == 26 && current == 26 || previous == 26 && previousP
              == 26) {
28          return 0;
29        }
30        return charFrequencyTriple[previousP, previous, current];
31      }
32      //Incializa la tabla con el texto dado
33      public void initCharFrequencyTripleTable(String s) {
34        int total = 0;
35        for (int i = 0; i < s.Length; i++) {
36          if (i > 1) {
37
38            int current = Helper.getIndex(s[i]);
39            int previous = Helper.getIndex(s[i - 1]);
40            int previousP = Helper.getIndex(s[i - 2]);
41            if (previous != -1 && current != -1 && previousP != -1) {
42              if ((previousP != 26 || previous != 26) && (previous != 26 ||
                    current != 26)) {
43                total++;
44                charFrequencyTriple[previousP, previous, current]++;
45              }
46
47            }
48          }
49        }
50        for (int i = 0; i < 27; i++) {
51          for (int j = 0; j < 27; j++) {
52            for (int k = 0; k < 27; k++) {
53              charFrequencyTriple[i, j, k] = charFrequencyTriple[i, j, k] / total
                    * 100;
54              if (charFrequencyTriple[i, j, k] > max) {
55                max = charFrequencyTriple[i, j, k];
56              }
57            }
58          }
59        }
60      }
61    }
62  }
```

BIBLIOGRAFÍA

[1] S. Russell and P. Norvig, *Artificial intelligence a modern approach*. Pearson Education, 2010. 1.1.2

[2] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, 1998. 2.1.2

[3] S. Luke, *Essentials of Metaheuristics*. Lulu, 2013. 2.1.2

[4] T. V. Mathew, "Genetic algorithm," Master's thesis, Indian Institute of Technology Bombay. 2.4

[5] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., 2008. 3.1.1

[6] A. Valsecchi and S. Damas, "An image registration approach using genetic algorithms," Master's thesis, University of Jaen, 2012. 3.2.2

[7] H. T. T. Chi Kin Chow and T. Lee, "Surface registration using a dynamic genetic algorithm," Master's thesis, The Chinese University of Hong Kong, 2003. 3.2.7

[8] M. Johansson, "Image registration with simulated annealing and genetic algorithms," Master's thesis, Royal Institute of Technology, 2006. 3.2.7

[9] B. Delman, "Decrypting substitution ciphers with genetic algorithms," Master's thesis, Rochester Institute of Technology, 2004. 4.1

[10] J. Brownbridge, "Genetic algorithms in cryptography," Master's thesis, University of Cape Town, 2007. 4.2.3

[11] A. J. Bagnall, "The applications of genetic algorithms in cryptanalysis," Master's thesis, University of East Anglia, 1996. 4.2.5

[12] D. A. Coley, *An Introduction to Genetic Algorithms for Scientists and Engineers*. World Scientific Publishing Co., Inc., 1998. 5.1