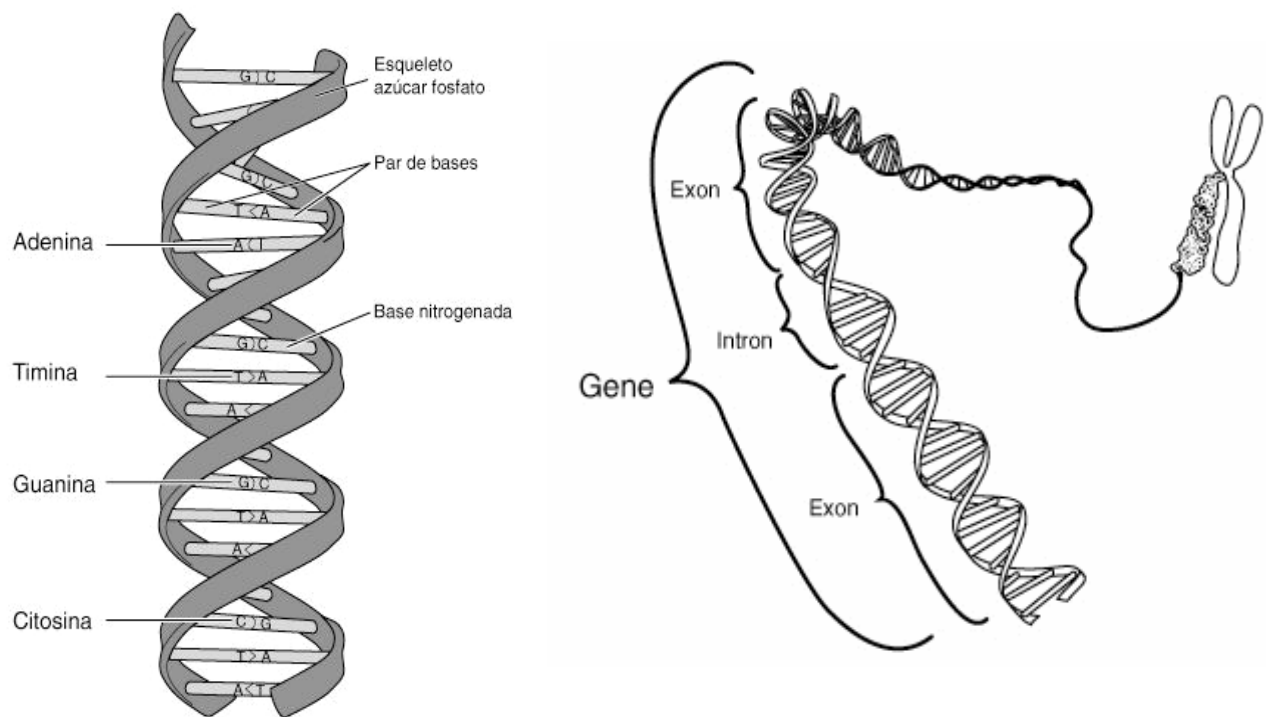


**Trabajo Práctico N<sup>ro</sup> 3 : Implementación de Algoritmos Genéticos**

“Optimización en un proceso de corte de cañerías”



**Materia: Introducción a la Inteligencia Artificial – UTN FRBA**

**Profesor: Dr. Claudio Verrastro**

**Ayudante: Ing. Juan Carlos Gómez**

**Alumno: Gustavo Damián Gil**

## **Índice de contenidos:**

Enunciado de la Problemática .....	3
Definición de Esquema .....	4
Cimientos para construir un algoritmo genético general .....	5
Esquema gráfico de los principales vectores y matrices intervinientes .....	5
¿Como ejecutar el programa? .....	7

### **Anexo:**

¿Que es un algoritmo genético?.....	9
Codificación de las variables .....	9
Algoritmo genético propiamente dicho .....	10
Evaluación y selección .....	10
Crossover .....	11
Mutación .....	13
Otros operadores .....	13
Aplicando operadores genéticos .....	14
Buenas prácticas en los Algoritmos Genéticos .....	14
¿Cómo saber si es posible usar un Algoritmo Genético? .....	15
Algunas aplicaciones de los Algoritmos Genéticos .....	16

## **Enunciado de la Problemática:**

Dada una cantidad (n) de tramos de caño, de longitudes  $l_1, l_2, \dots, l_n$ , necesarios para una obra, se trata de minimizar el desperdicio que se genera al realizar los cortes de los caños, que vienen en tiras de una longitud determinada ( $L_{\max}$ ).

Consideraciones varias:

1. Aún cuando los tramos sean de igual longitud, se los considerará tramos diferentes.
2. Aún cuando haya soluciones inválidas, tales como que de una tira de caño se obtienen más tramos de lo que su longitud permite, esto se tendrá en cuenta en una alinealidad de la función de fitness.
3. El hecho de que haya soluciones que compitan entre sí, debería ser resuelto por el algoritmo genético. Como en el caso de equivalencia de que una combinación de tramos se obtenga de una tira u otra. Las tiras son indiferentes.
4. En esta representación, no importa el orden de los tramos dentro de la tira.

### **Definición de la representación:**

Cada individuo será un string, donde cada posición se corresponde con una longitud determinada, y su valor indicará de que tira se obtiene ( $T_1, T_2, \dots, T_m$ ). De la forma:

$l_1$	$l_2$	$l_3$	$l_4$	$l_5$	$l_6$	$l_7$	$l_8$	$l_9$	$l_{10}$	$l_{11}$	$l_{12}$	$l_{13}$	$l_{14}$	$l_{15}$	$l_{16}$	$l_{17}$	$l_{18}$	$l_{19}$	$l_{20}$
$T_1$	$T_2$	$T_2$	$T_1$	$T_1$	$T_3$	$T_2$	$T_3$	$T_2$	$T_3$	$T_1$	$T_1$	$T_3$	$T_1$	$T_2$	$T_2$	$T_1$	$T_2$	$T_3$	$T_3$

Donde la fila de arriba indica el tramo de caño necesario ordenados según sus longitudes y la de abajo de cual tira se obtiene.

La identificación de cada tira podría ser binaria, con una cantidad de dígitos que posibilite la solución al problema. Caso contrario se puede definir un alfabeto de tantos símbolos como tiras posibles. Es un valor a estimar.

### **Función de fitness:**

Se trata de minimizar la cantidad de tiras a utilizar junto a la longitud total del desperdicio.

Se calculará la función de la siguiente manera:

(sólo para las tiras utilizadas, ya que puede haber tiras de las que no se haga ningún corte)

La función será una suma de la cantidad de tiras y términos obtenidos por una función del desperdicio (d) obtenido para cada una de las tiras.

$$Ff = \text{Cantidad de tiras} + \sum F(d_x)$$

Para cada tira se suman las longitudes de los tramos que de ella se sacan.

$$T_1 = \sum (l_1, l_4, l_5, l_{11}, l_{12}, l_{14}, l_{17})$$

$$T_2 = \sum (l_2, l_3, l_7, l_9, l_{15}, l_{16}, l_{18})$$

$$T_3 = \sum (l_6, l_8, l_{10}, l_{13}, l_{19}, l_{20})$$

La función se hace alineal, de manera de penalizar aquellas soluciones que saquen más de la tira de lo que la tira tiene (soluciones no posibles).

Definiendo desperdicio como

$$d_x = (L_{\max} - T_x)$$

ver que  $d_x$  está definida entre

$$[(L_{\max} - \sum l_x), L_{\max}]$$

Si  $d_x \geq 0$  podría ser  $F(d_x) = d_x / L_{\max}$  (está normalizada a la longitud de la tira)

Si  $d_x < 0$  podría ser  $F(d_x) = (-d_x / L_{\max}) + 1$

Así planteado el problema, su representación y función de aptitud, se puede resolver con los operadores de selección, cruce y mutación definidos y por lo tanto vale el teorema del esquema.

Recordemos la definición de esquema:

“Grupo de Genes de un individuo, que tienen relevancia en la aptitud óptima.”

Parámetros de entrada del archivo e incorporación de los datos:

Importaremos un archivo de texto con los datos separados por " ; " que en su primer fila contiene la longitud de los listones comprados, en nuestro caso Lmax. Las resultantes filas (no se sabe cuantas pueden ser) tienen dos números enteros, el primero es la cantidad de caños cortados necesarios de la longitud es especificada por el segundo numero.

El nombre del archivo de entrada siempre será: agXX.txt (donde XX es un numero, por ej. ag01.txt)  
El archivo de salida a generar deberá ser: agXXGDG.txt (siendo GDG las iniciales del alumno).  
Este archivo de salida informara la Cantidad de tiras utilizadas y la suma total de los desperdicios.

---

Para realizar la práctica propuesta, se planteó por mi parte su realización en el lenguaje de programación MATLAB, ya que comprendí que la esencia de un Algoritmo Genético es matricial y muy vectorizado, así que dicho lenguaje de programación resultaba muy adecuado.

Según las recomendaciones del Ing. Gómez, se pensó el algoritmo lo mas simplificado posible, pensando en cantidades de cortes inferiores a las 1000 como se propuso inicialmente, y luego se lo expandió. El presente algoritmo puede ser mas generalizado y optimizado, en una posible implementación real, pero así como se lo presenta funciona modestamente bien.


Ahora describiré brevemente los *cimientos para construir un algoritmo genético general*, como el desarrollado:





















- 1- Definir la forma de representación de cada individuo (una posible solución).
- 2- Generar una población de individuos en forma aleatoria.
- 3- Evaluar la población, y obtener el mérito o aptitud de cada individuo (fitness).
- 4- Obtener el mérito medio o aptitud de la población.
- 5- En base a la aptitud promedio y las aptitudes individuales, podemos seleccionar a los individuos mas aptos para su reproducción, y a los menos aptos para su reemplazo o eventual mutación, producto de la evolución de la población para alcanzar una solución.
- 6- Realizamos un LOOP al paso tres “n” veces (una cantidad definida), o hasta que la población no pueda evolucionar mas (se aconseja siempre limitar la cantidad de iteraciones).

Como particularidades de mi implementación mencionaré:

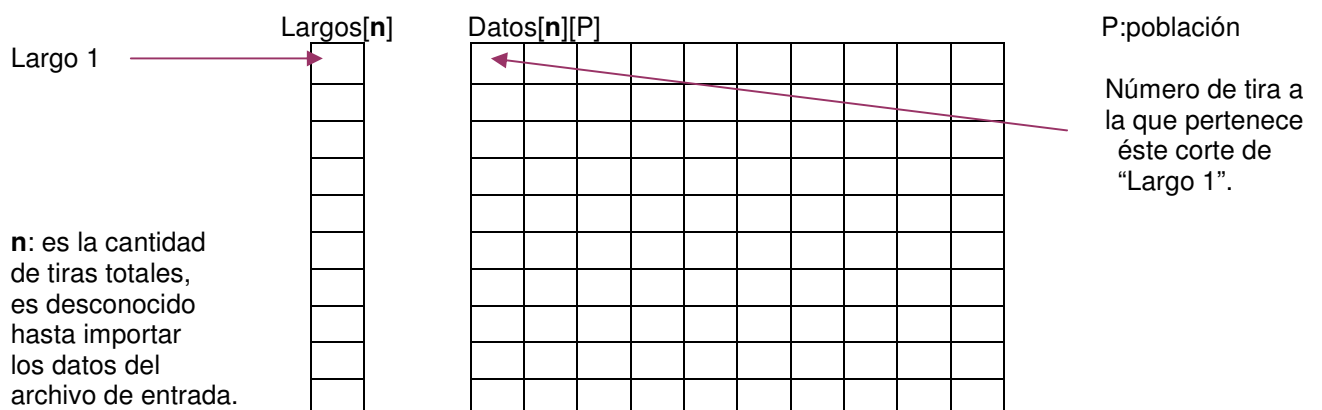
- La población de soluciones, siempre mantiene de un tamaño constante.
- Los hijos generados por los individuos mas aptos, reemplazan en la siguiente generación a los individuos menos aptos, sin excepción.
- La posición física del almacenamiento de la información (los genes), no afecta al desarrollo evolutivo, ya que se emplean máscaras de corte, para realizar el apareamiento (intercambio de material genético).
- La selección surge de un umbral definido, con respecto a la aptitud poblacional promedio.

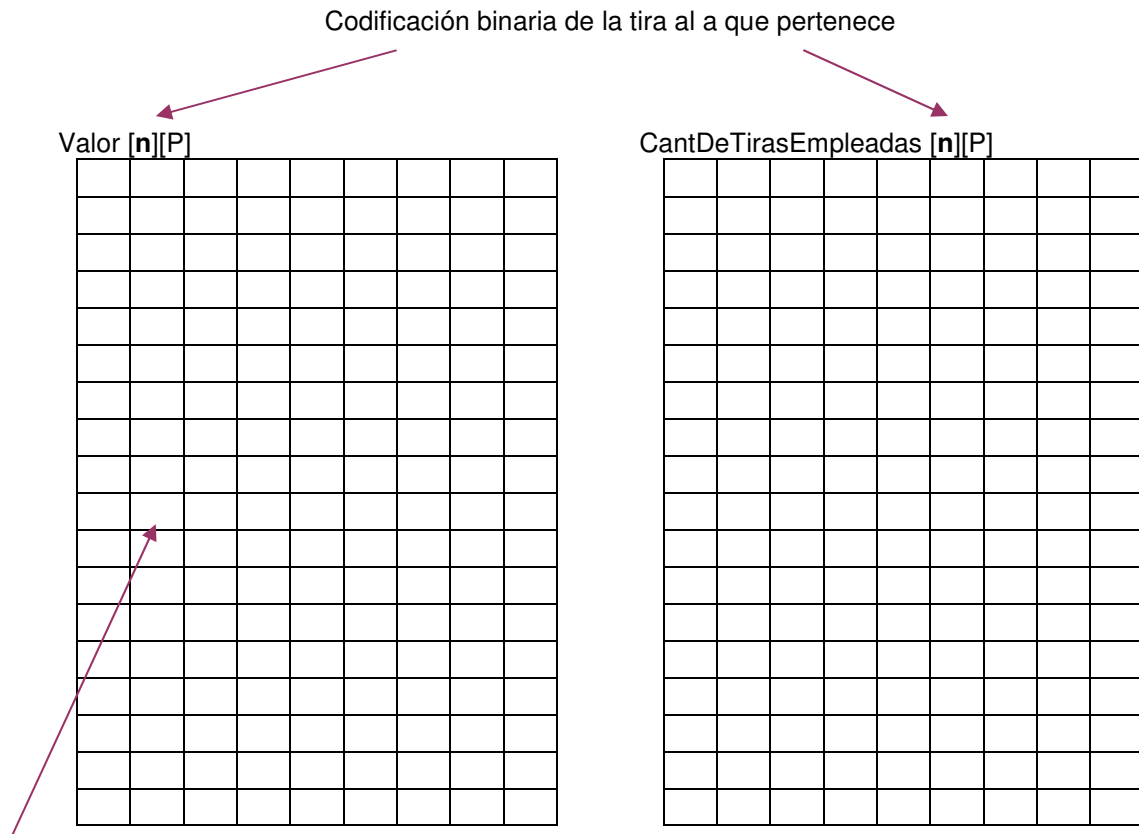
Tamaño de las variables empleadas en la implementación (nota muchas de ellas pudieron ser liberadas después de su utilización, pero no se realizó por razones didácticas).

Name	Size
 AcumulaDesecho	1x100
 AcumulaTirasEmpleadas	1x100
 AptiProm	1x1
 Aptitud	1x100
 Aptitud_Decreciente	1x100
 CantDeTirasEmpleadas	256x100
 CantDeTirasTotales	1x1
 CantItera	1x1
 Cantidades	9x1
 Datos	256x100
 Evaluacion_Apareos	1x100
 Evaluacion_AptiProm	1x100
 Evaluacion_Mutaciones	1x100
 ExtractA	256x6
 ExtractB	256x6
 Flag	1x6
 Flag1	1x4
 Indice_Para_Reemplazar	1x6
 Indices_Para_Aparear	1x0
 Iteracion	1x100
 Largos	256x1

Name	Size
 Largos	256x1
 Lmax	1x1
 Longitudes	9x1
 Mask	256x1
 Not_Mask	256x1
 P	1x1
 ProbaMuta	1x1
 String	1x11
 UmbralApareo	1x1
 Valor	256x100
 a1	1x1
 a2	1x1
 acumulador	256x1
 c1	1x1
 c2	1x1
 contador	1x1
 i	1x12
 iteraciones	1x1
 j	1x1
 m	1x1

Esquema gráfico de los principales vectores y matrices intervinientes en el algoritmo genético:





El contenido del Valor, es la sumatoria de las longitudes de caños empleados por listón. Al finalizar el proceso evolutivo en la última generación, el contenido mayoritario de ésta matriz deben ser valores en cero (que indican que ése listón no se utilizó) y valores Lmax (máximo aprovechamiento del listón.)

Marca[P]

--	--	--	--	--	--	--	--

## ¿Como ejecutar el programa?

Si bien se desarrolló parte en una versión de Matlab 6.5 y 7.4, por el tipo de comandos empleados se supone que pude funcionar correctamente desde la versión 5 en adelante, pero no fue constatado.

Se debe colocar en un directorio el programa “ag8.m”, junto con el archivo de entrada de información que es del tipo mencionado en el enunciado, cabe aclarar que el archivo entregado como respuesta también se almacenará en dicho directorio.

Luego iniciamos el Matlab, y redirigimos el path del Current Directory a dicha carpeta.



Finalmente ejecutamos el programa haciendo:

```
>> ag8
```

Y el programa nos responde con lo siguiente:

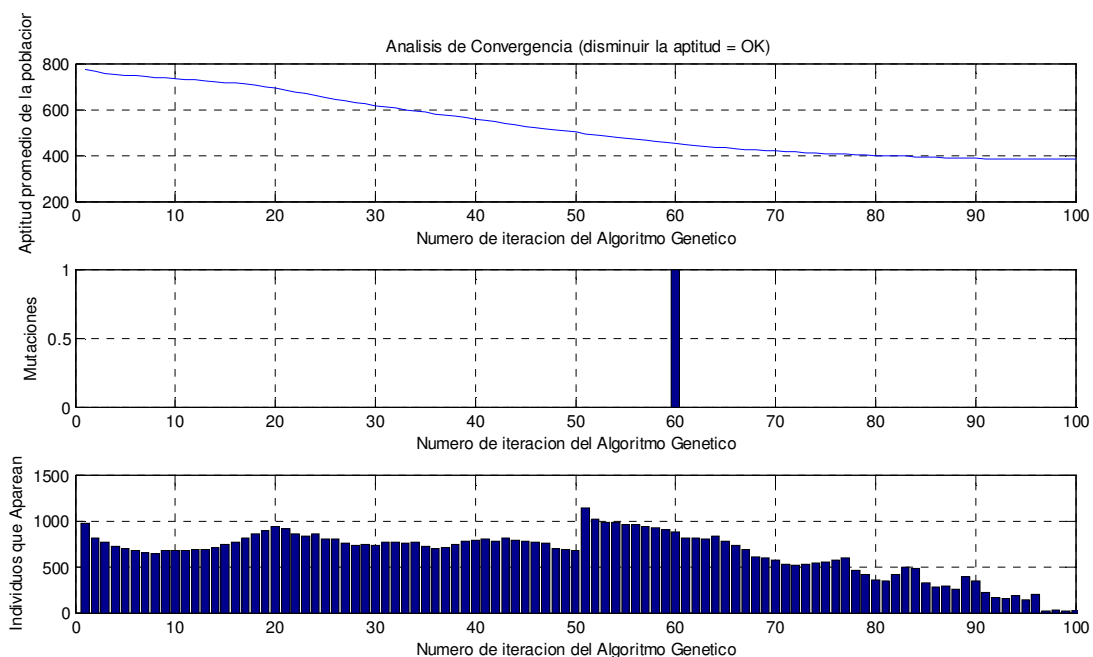
**Este es un programa para optimizar la compra de caños.  
Por favor, verifique que el archivo con los datos se encuentre en  
el directorio de trabajo.**

**Ingrese el nombre del archivo con datos entre apostrofes (comillas  
simples) = 'ag05.txt'**

Donde ingresamos el nombre del archivo a analizar, que en nuestro caso fue “ 'ag05.txt' ”, y luego nos consulta como queremos modificar su estructura de funcionamiento, y nos indica unos valores mínimos por si desconocemos el funcionamiento del mismo para orientarnos.

Una vez definidos los parámetros de funcionamiento, presenta la solución obtenida junto con una gráfica del comportamiento del algoritmo, para tener un conocimiento mas acabado de cómo realizó la evolución hasta encontrar la solución.

Para nuestro caso particular empleamos una Población=5000 y Iteraciones=100 debido a la gran cantidad de cortes requeridos, si nos interesa que el programa explore mas posibilidades en forma paralela, debemos incrementar la Población.



El primer gráfico indica el valor de la aptitud promedio de la población, a medida de las iteraciones o generaciones, y siempre debe ser decreciente. Cuánto menor sea el valor de aptitud promedio, es una mejor solución, pero eso depende de lo que se busque optimizar.

El segundo gráfico indica la cantidad de mutaciones que se produjeron, en éste caso se presentó una mutación en la generación 60, pero en la gran mayoría de los casos, no se presentan mutaciones.

El tercero, indica la cantidad de individuos que se aparearon o cruzaron, para transmitir sus características a la próxima generación. El hecho de que a partir de la generación 97 no se produjeran muchos apareamientos, nos indica que la elección de 100 iteraciones es adecuada.

Si hubiéramos elegido 200 iteraciones, seguramente las últimas 100 o 90 generaciones, no se hubieran producido cuzas, lo que significa que se obtuvo una buena solución antes de lo esperado.

Finalmente el resultado de la solución puede visualizarse tanto en pantalla como en el archivo de salida, el de éste caso fue:

**Este es un programa para optimizar la compra de caños.**

**Por favor, verifique que el archivo con los datos se encuentre en el directorio de trabajo.**

**Ingrese el nombre del archivo con datos entre apostrofes (comillas simples) = 'ag05.txt'**

**Ahora podremos experimentar modificando la estructura de funcionamiento de nuestro algoritmo:**

**Ingrese la cantidad de Individuos de la Poblacion de posibles soluciones (minimo 50)=5000**

**P =**  
**5000**

**Ingrese la cantidad de Iteraciones o Repeticiones del algoritmo (minimo 30)=100**

**CantItera =**  
**100**

**Desechos acumulados de caños (DespA) :**  
**6.8750**

**Desechos irrealizables de caños (DespB) :**  
**-2.8750**

**Combinaciones Perfectas (DespC) :**  
**125**

**Tiras de caños comprados empleados:**  
**64**

**NOTA: miranfo la matriz Datos, subindicada por i(1), junto con el vector Largos, podemos ver como realizar los cortes.**

**>>**

Finalmente, conociendo que la solución perfecta de éste problema es 60 caños empleados sin desperdicios, resulta que la estimación de 64 caños es buena, igualmente podemos notar que si sumamos los DespA y DespB y se lo restamos a la estimación de “Tiras de caños comprados empleados”, veremos que casualmente nos da la solución perfecta. Esto no siempre sucede, pero si se presenta mayoritariamente, lo que nos indica que el sistema estima en forma adecuada.



## ¿Que es un algoritmo genético?

Primero nos deberíamos preguntar, ¿cómo logra la naturaleza crear seres cada vez más perfectos? (aunque, como se ha visto, esto no es totalmente cierto, o en todo caso depende de qué entienda uno por *perfecto*).

La respuesta se puede hallar haciendo pequeños modelos de la naturaleza, que tuvieran alguna de sus características, y ver cómo funcionan, para luego extrapolar conclusiones.

Después de esto, se pudo concluir que los *algoritmos genéticos* son métodos sistemáticos para la resolución de problemas de búsqueda y optimización, que aplican a estos los mismos métodos de la evolución biológica: selección basada en la población, reproducción sexual y mutación.

En un algoritmo genético, tras parametrizar el problema en una serie de variables,  $(x_1, \dots, x_n)$  que se codifican en un cromosoma. Todos los operadores utilizados por un algoritmo genético, se aplicarán sobre estos cromosomas, o sobre poblaciones de ellos. En el algoritmo genético va implícito el método para resolver el problema; son solo parámetros de tal método los que están codificados, a diferencia de otros algoritmos evolutivos como la programación genética. Hay que tener en cuenta que un algoritmo genético es independiente del problema, lo cual lo hace un algoritmo *robusto*, por ser útil para cualquier problema, pero a la vez *débil*, pues no está especializado en ninguno.

Las soluciones codificadas en un cromosoma *compiten* para ver cuál constituye la mejor solución (aunque no necesariamente la mejor de todas las soluciones posibles). El *ambiente*, constituido por las otras camaradas soluciones, ejercerá una presión selectiva sobre la población, de forma que sólo los mejor adaptados (aquellos que resuelvan mejor el problema) sobrevivan o leguen su material genético a las siguientes generaciones, igual que en la evolución de las especies. La diversidad genética se introduce mediante mutaciones y reproducción sexual.

En la Naturaleza lo único que hay que optimizar es la supervivencia, y eso significa a su vez maximizar diversos factores y minimizar otros. Un algoritmo genético, sin embargo, se usará habitualmente para optimizar sólo una función, no diversas funciones relacionadas entre sí simultáneamente. La optimización que busca diferentes objetivos simultáneamente, denominada multimodal o multiobjetivo, también se suele abordar con un algoritmo genético especializado.

*Por lo tanto, un algoritmo genético consiste en lo siguiente: hallar de qué parámetros depende el problema, codificarlos en un cromosoma, y se aplican los métodos de la evolución: selección y reproducción sexual con intercambio de información y alteraciones que generan diversidad.*

## Codificación de las variables

Los algoritmos genéticos requieren que el conjunto se codifique en un *cromosoma*. Cada cromosoma tiene varios genes, que corresponden a sendos parámetros del problema. Para poder trabajar con estos genes en la computadora, es necesario codificarlos en una *cadena*, es decir, una ristra de símbolos (números o letras) que generalmente va a estar compuesta de 0s y 1s.

Por ejemplo, en esta cadena de bits, el valor del parámetro  $p1$  ocupará las posiciones 0 a 2, el  $p2$  las 3 a 5, etcétera. El número de bits usado para cada parámetro dependerá de la precisión que se quiera en el mismo.

Hay otras codificaciones posibles, usando alfabetos de diferente cardinalidad (número de elementos constitutivos); sin embargo, uno de los resultados fundamentales en la teoría de algoritmos

genéticos, el *teorema de los esquemas*, afirma que la codificación óptima, es decir, aquella sobre la que los algoritmos genéticos funcionan mejor, es aquella que tiene un alfabeto de cardinalidad 2.

La mayoría de las veces, una codificación correcta es la clave de una buena resolución del problema. Generalmente, la regla heurística que se utiliza es la llamada *regla de los bloques de construcción*, es decir, parámetros relacionados entre sí deben de estar cercanos en el cromosoma. Por ejemplo, si queremos codificar los pesos de una red neuronal, una buena elección será poner juntos todos los pesos que salgan de la misma neurona de la capa oculta.

Se puede ser bastante creativo con la codificación del problema, teniendo siempre en cuenta la regla anterior. Esto puede llevar a usar cromosomas bidimensionales, o tridimensionales, o con relaciones entre genes que no sean puramente lineales de vecindad. En algunos casos, cuando no se conoce de antemano el número de variables del problema, caben dos opciones: codificar también el número de variables, fijando un número máximo, o bien, lo cual es mucho más natural, crear un cromosoma que pueda variar de longitud. Para ello, claro está, se necesitan operadores genéticos que alteren la longitud.

### Algoritmo genético propiamente dicho

Para comenzar la competición, se generan aleatoriamente una serie de cromosomas. El algoritmo genético procede de la forma siguiente:

1. Evaluar la puntuación (*fitness*) de cada uno de los genes.
2. Permitir a cada uno de los individuos reproducirse, de acuerdo con su puntuación.
3. Emparejar los individuos de la nueva población, haciendo que intercambien material genético, y que eventualmente alguno de los bits de un gen se vea alterado debido a una *mutación* espontánea.

Cada uno de los pasos consiste en una actuación sobre las cadenas de bits, es decir, la aplicación de un *operador* a una cadena binaria. Se les denominan, por razones obvias, *operadores genéticos*, y hay tres principales: *selección*, *crossover* o apareamiento, y *mutación*; aparte de otros operadores genéticos no tan comunes.

Un algoritmo genético tiene también una serie de parámetros que se tienen que fijar:

- **Tamaño de la población:** debe de ser suficiente para garantizar la diversidad de las soluciones, y además tiene que crecer más o menos con el número de bits del cromosoma, aunque nadie ha aclarado cómo tiene que hacerlo. Nosotros no lo haremos.
- **Condición de terminación:** lo más habitual es que la condición de terminación sea la convergencia del algoritmo genético o un número prefijado de generaciones.

### Evaluación y selección

Durante la evaluación, se decodifica el gen, convirtiéndose en una serie de parámetros de un problema, se halla la solución del problema a partir de esos parámetros, y se le da una puntuación a esa solución en función de lo cerca que esté de la mejor solución. A esta puntuación se le llama *fitness* o aptitud.

Por ejemplo, supongamos que queremos hallar el máximo una parábola invertida con el máximo en  $x=1$ . En este caso, el único parámetro del problema es la variable  $x$ . La optimización consiste en hallar un  $x$  tal que  $F(x)$  sea máximo. Crearemos, pues, una población de cromosomas, cada uno de los cuales contiene una codificación binaria del parámetro  $x$ . Lo haremos de la forma siguiente: cada

byte, cuyo valor está comprendido entre 0 y 255, se transformará para ajustarse al intervalo  $[-1,1]$ , donde queremos hallar el máximo de la función.

Valor binario	Decodificación	Evaluación $f(x)$
10010100	21	0,9559
10010001	19	0,9639
101001	-86	0,2604
1000101	-58	0,6636

El fitness determina los cromosomas que se van a reproducir, y aquellos que se van a eliminar, pero hay varias formas de considerarlo para seleccionar la población de la siguiente generación:

- Usar el orden, o rango, y hacer depender la probabilidad de permanencia, o evaluación de la posición en el orden.
- Aplicar una operación al fitness; como por ejemplo el *escalado sigma*. La reproducción se basa en una comparación estadística de la aptitud media de la población, para evitar la convergencia prematura hacia un óptimo local.
- En algunos casos, el fitness no es una sola cantidad, sino diversos números, que tienen diferente consideración. Basta con que tal fitness forme un orden parcial, es decir, que se puedan comparar dos individuos y decir cuál de ellos es mejor. Esto suele suceder cuando se necesitan optimizar varios objetivos.

Una vez evaluado el fitness, se tiene que crear la nueva población teniendo en cuenta que los *buenos* rasgos de los mejores, se transmitan a esta. Está la selección, y la consiguiente reproducción:

- *Basado en el rango*: en este esquema se mantiene un porcentaje de la población, generalmente la mayoría, para la siguiente generación. Se coloca toda la población por orden de fitness, y los  $M$  menos dignos son eliminados y sustituidos por la descendencia de alguno de los  $M$  mejores con algún otro individuo de la población. A este esquema se le pueden aplicar otros criterios; por ejemplo, se crea la descendencia de uno de los mejores, y esta sustituye al más parecido entre los perdedores. Esto se denomina *crowding*. Una variante de este es el muestreo estocástico universal, que trata de evitar que los individuos con más fitness copen la población; en vez de dar la vuelta a una ruleta con una ranura, da la vuelta a la ruleta con  $N$  ranuras, tantas como la población; de esta forma, la distribución estadística de descendientes en la nueva población es más parecida a la real.
- *Rueda de ruleta*: se crea un *pool* genético formado por cromosomas de la generación actual, en una cantidad proporcional a su fitness. Si la proporción hace que un individuo domine la población, se le aplica alguna operación de escalado. Dentro de este *pool*, se cogen parejas aleatorias de cromosomas y se emparejan, sin importar incluso que sean del mismo progenitor (para eso están otros operadores, como la mutación). Hay otras variantes: por ejemplo, en la nueva generación se puede incluir el mejor representante de la generación actual. En este caso, se denomina método *elitista*.
- *Selección de torneo*: se escogen aleatoriamente un número  $T$  de individuos de la población, y el que tiene puntuación mayor se reproduce, sustituyendo su descendencia al que tiene menor puntuación.

## Crossover

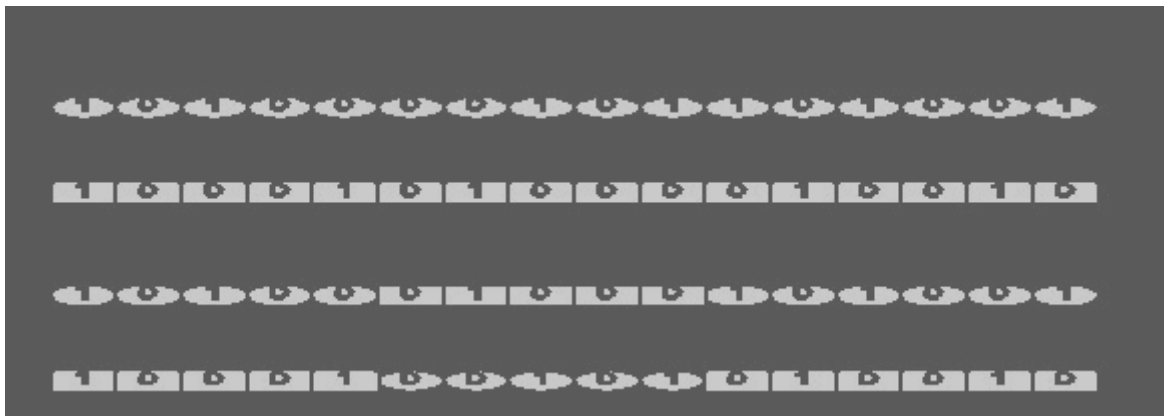
Consiste en el intercambio de material genético entre dos cromosomas (apareamiento). El *crossover* es el principal operador genético, hasta el punto que se puede decir que no es un algoritmo genético si no tiene *crossover*, sin embargo, puede serlo perfectamente sin mutación.

Para aplicar el crossover, entrecruzamiento o recombinación, se escogen aleatoriamente dos miembros de la población. No pasa nada si se emparejan dos descendientes de los mismos padres; ello garantiza la perpetuación de un individuo con buena puntuación (algo parecido ocurre en la cría de ganado, llamada *inbreeding*, destinada a potenciar ciertas características frente a otras). Sin embargo, si esto sucede demasiado a menudo, puede crear problemas: toda la población puede aparecer dominada por los descendientes de algún gen, que puede tener caracteres no deseados. Esto se suele denominar en otros métodos de optimización *atranque en un mínimo local*, y es uno de los principales problemas con los que se enfrentan los que aplican algoritmos genéticos.

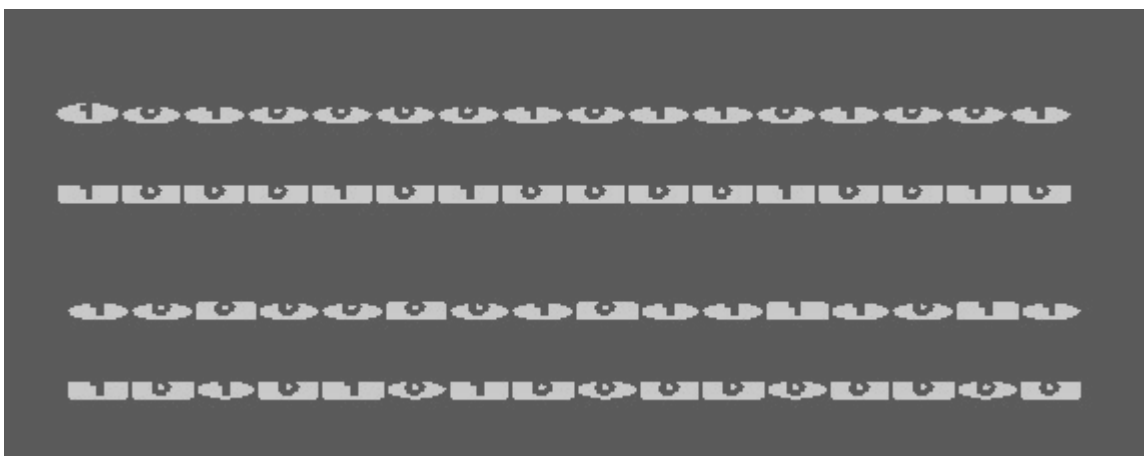
En cuanto al teorema de los esquemas, se basa en la noción de *bloques de construcción*. Una buena solución a un problema está constituida por unos buenos bloques, igual que una buena máquina está hecha por buenas piezas. El crossover es el encargado de mezclar bloques buenos que se encuentren en los diversos progenitores, y que serán los que den a los mismos una buena puntuación. La presión selectiva se encarga de que sólo los buenos bloques se perpetúen, y poco a poco vayan formando una buena solución. El *teorema de los esquemas* viene a decir que la cantidad de *buenos bloques* se va incrementando con el tiempo de ejecución de un algoritmo genético, y es el resultado teórico más importante en algoritmos genéticos.

El intercambio genético se puede llevar a cabo de muchas formas, pero hay dos grupos principales:

- *Crossover n-puntos*: los dos cromosomas se cortan por n puntos, y el material genético situado entre ellos se intercambia. Lo más habitual es un crossover de un punto o dos.



- *Crossover uniforme*: se genera un patrón aleatorio de 1s y 0s, y se intercambian los bits de los dos cromosomas que coincidan donde hay un 1 en el patrón. O bien se genera un número aleatorio para cada bit, y si supera una determinada probabilidad se intercambia ese bit entre los dos cromosomas.



- *Crossover especializados*: en algunos problemas, aplicar aleatoriamente el crossover da lugar a cromosomas que codifican soluciones inválidas; en este caso hay que aplicar el crossover de forma que genere siempre soluciones válidas.

## Mutación

En la Evolución, una mutación es un suceso bastante poco común (sucede aproximadamente una de cada mil replicaciones). En la mayoría de los casos las mutaciones son letales, pero en promedio, contribuyen a la diversidad genética de la especie. En un algoritmo genético tendrán el mismo papel, y la misma frecuencia.

Una vez establecida la frecuencia de mutación, por ejemplo, uno por mil, se examina cada bit de cada cadena cuando se vaya a crear la nueva criatura a partir de sus padres (normalmente se hace de forma simultánea al crossover). Si un número generado aleatoriamente está por debajo de esa probabilidad, se cambiará el bit (es decir, de 0 a 1 o de 1 a 0). Si no, se dejará como está. Dependiendo del número de individuos que haya y del número de bits por individuo, puede resultar que las mutaciones sean extremadamente raras en una sola generación.

No hace falta decir que no conviene abusar de la mutación. Es cierto que es un mecanismo generador de diversidad, y por tanto, la solución cuando un algoritmo genético está estancado, pero también es cierto que reduce el algoritmo genético a una búsqueda aleatoria. Siempre es más conveniente usar otros mecanismos de generación de diversidad, como aumentar el tamaño de la población, o garantizar la aleatoriedad de la población inicial.

Este operador, junto con la anterior y el método de selección de ruleta, constituyen un *algoritmo genético simple*.

## Otros operadores

No se usan en todos los problemas, sólo en algunos, y en principio su variedad es infinita. Generalmente son operadores que exploran el espacio de soluciones de una forma más ordenada, y que actúan más en las últimas fases de la búsqueda, en la cual se pasa de soluciones "casi buenas" a "buenas" soluciones.

### - Cromosomas de longitud variable

Hasta ahora se han descrito cromosomas de longitud fija, donde se conoce de antemano el número de parámetros de un problema. Pero hay problemas en los que esto no sucede. Por ejemplo, en un problema de clasificación, donde dado un vector de entrada, queremos agruparlo en una serie de clases, podemos no saber siquiera cuantas clases hay. O en diseño de redes neuronales, puede que no se sepa (de hecho, nunca se sabe) cuántas neuronas se van a necesitar. Por ejemplo, en un perceptrón hay reglas que dicen cuantas neuronas se deben de utilizar en la capa oculta; pero en un problema determinado puede que no haya ninguna regla heurística aplicable; tendremos que utilizar los algoritmos genéticos para hallar el número óptimo de neuronas.

En estos casos, necesitamos dos operadores más: *añadir* y *eliminar*. Estos operadores se utilizan para añadir un gen, o eliminar un gen del cromosoma. La forma más habitual de añadir es *duplicar* uno ya existente, el cual sufre mutación y se añade al lado del anterior. En este caso, los operadores del algoritmo genético simple (selección, mutación, crossover) funcionarán de la forma habitual, salvo, que sólo se hará crossover en la zona del cromosoma de menor longitud.

Estos operadores permiten, además, crear un *algoritmo genético de dos niveles*: a nivel de cromosoma, y a nivel de gen. Supongamos que, en un problema de clasificación, hay un gen por clase. Se puede asignar una puntuación a cada gen en función del número de muestras que haya clasificado correctamente. Al aplicar estos operadores, se duplicarán los alelos (una codificación determinada del gen) con mayor puntuación, y se eliminarán aquellos que hayan obtenido menor puntuación.

### - Operadores de nicho (ecológico)

Otros operadores importantes son los operadores de *nicho*. Estos operadores están encaminados a mantener la diversidad genética de la población, de forma que cromosomas similares sustituyan sólo a cromosomas similares, y son especialmente útiles en problemas con muchas soluciones; un algoritmo genético con estos operadores es capaz de hallar todos los máximos, dedicándose cada especie a un máximo. Más que operadores genéticos, son formas de enfocar la selección y la evaluación de la población.

Una de las formas de llevar esto a cabo ya ha sido nombrada, la introducción del *crowding* (se crea la descendencia de uno de los mejores, y esta sustituye al más parecido entre los perdedores). Otra forma es introducir una función de compartición o *sharing*, que indica cuán similar es un cromosoma al resto de la población. La puntuación de cada individuo se dividirá por esta función de compartición, de forma que se facilita la diversidad genética y la aparición de individuos diferentes.

También se pueden restringir los emparejamientos, por ejemplo, a aquellos cromosomas que sean similares. Para evitar las malas consecuencias del inbreeding, que suelen aparecer en poblaciones pequeñas.

### - Operadores especializados

En una serie de problemas hay que restringir las nuevas soluciones generadas por los operadores genéticos, pues no todas las soluciones generadas van a ser válidas, sobre todo en los problemas con restricciones. Por ello, se aplican operadores que mantengan la estructura del problema. Otros operadores son simplemente generadores de diversidad, pero la generan de una forma determinada:

- **Zap:** en vez de cambiar un solo bit de un cromosoma, cambia un gen completo.
- **Creep:** este operador aumenta o disminuye en 1 el valor de un gen; sirve para cambiar suavemente y de forma controlada los valores de los genes.
- **Transposición:** similar al crossover y a la recombinación genética, pero dentro de un solo cromosoma; dos genes intercambian sus valores, sin afectar al resto del cromosoma. Similar a este es el operador de **eliminación-reinserción**, en el que un gen cambia de posición con respecto a los demás.

### Aplicando operadores genéticos

En toda ejecución de un algoritmo genético hay que decidir con qué frecuencia se va a aplicar cada uno de los algoritmos genéticos; en algunos casos, como en la mutación o el crossover uniforme, se debe añadir algún parámetro adicional, que indique con qué frecuencia se va a aplicar dentro de cada gen del cromosoma. La frecuencia de aplicación de cada operador estará en función del problema; teniendo en cuenta los efectos de cada operador, tendrá que aplicarse con cierta frecuencia o no. Generalmente, la mutación y otros operadores que generen diversidad se suele aplicar con poca frecuencia; la recombinación se suele aplicar con frecuencia alta.

En general, la frecuencia de los operadores no varía durante la ejecución del algoritmo, pero hay que tener en cuenta que cada operador es más efectivo en un momento de la ejecución. Por ejemplo, al principio, en la fase denominada de *exploración*, los más eficaces son la mutación y la recombinación; posteriormente, cuando la población ha convergido en parte, la recombinación no es útil, pues se está trabajando con individuos bastante similares, y es poca la información que se intercambia. Sin embargo, si se produce un estancamiento, la mutación tampoco es útil, pues está reduciendo el algoritmo genético a una búsqueda aleatoria; y hay que aplicar otros operadores. En todo caso, se pueden usar operadores especializados.

## Buenas prácticas en los Algoritmos Genéticos

Deja que la Naturaleza sea tu guía: dado que la mayoría de los problemas a los que se van a aplicar los algoritmos genéticos son de naturaleza no lineal, es mejor actuar como lo hace la naturaleza, aunque intuitivamente pueda parecer la forma menos acertada. *Si queremos desarrollar sistemas no lineales que busquen y aprendan, mejor que comencemos (como mínimo) imitando a sistemas que funcionan (Goldberg).* Y estos sistemas se hallan en la naturaleza.

Cuidado con el asalto frontal: a veces se plantea el problema de pérdida de diversidad genética en una población de cromosomas. Hay dos formas de resolver este problema: aumentar el ritmo de mutación, lo cual equivale a convertir un algoritmo genético en un algoritmo de búsqueda aleatoria, o bien introducir mecanismos como el *sharing*, por el cual el fitness de un individuo se divide por el número de individuos similares a él. Este segundo método, más parecido al funcionamiento de la naturaleza, en la cual cada individuo, por bueno que sea, tiene que compartir recursos con aquellos que hayan resuelto el problema de la misma forma, funciona mucho mejor. Otro caso que surge a menudo en los grupos de discusión de Usenet es el tratar de optimizar AGs mediante AGs; es mucho mejor tratar de entender el problema que acercarse a él de esta manera.

Respetar la criba de esquemas: para ello, lo ideal es utilizar alfabetos con baja cardinalidad (es decir, con pocas letras) como el binario.

No te fíes de la autoridad central: la Naturaleza actúa de forma distribuida, por tanto, se debe de minimizar la necesidad de operadores que "vean" a toda la población. Ello permite, además, una fácil paralelización del algoritmo genético. Por ejemplo, en vez de comparar el fitness de un individuo con todos los demás, se puede comparar sólo con los *vecinos*, es decir, aquellos que estén, de alguna forma, situados cerca de él.

## ¿Cómo saber si es posible usar un Algoritmo Genético?

La aplicación más común de los AG ha sido la solución de problemas de optimización, en donde han mostrado ser muy eficientes y confiables. Sin embargo, no todos los problemas pudieran ser apropiados para la técnica, y se recomienda en general tomar en cuenta las siguientes características del mismo antes de intentar usarla:

- Su espacio de búsqueda (sus posibles soluciones) debe de estar delimitado dentro de un cierto rango.
- Debe permitir definir una función de aptitud que nos indique que tan buena o mala es una cierta respuesta.
- Las soluciones deben codificarse de forma que resulte fácil de implementar en la computadora.

El primer punto es muy importante, y lo más recomendable es intentar resolver problemas que tengan espacios de búsqueda discretos, aunque éstos sean muy grandes. Sin embargo, también podrá

intentarse usar la técnica con espacios de búsqueda continuos, pero preferiblemente cuando exista un rango de soluciones relativamente pequeño.

## **Algunas aplicaciones de los Algoritmos Genéticos**

**Optimización:** Se trata de un campo especialmente abonado para el uso de los AG, por las características intrínsecas de estos problemas. No en vano fueron la fuente de inspiración para los creadores estos algoritmos.

Los AG se han utilizado en numerosas tareas de optimización, incluyendo la optimización numérica, y los problemas de optimización combinatoria.

**Programación automática:** Los AG se han empleado para desarrollar programas para tareas específicas, y para diseñar otras estructuras computacionales tales como el autómatas celular, y las redes de clasificación.

**Aprendizaje máquina:** Los AG se han utilizado también en muchas de estas aplicaciones, tales como la predicción del tiempo o la estructura de una proteína. Han servido asimismo para desarrollar determinados aspectos de sistemas particulares de aprendizaje, como pueda ser el de los pesos en una red neuronal, las reglas para sistemas de clasificación de aprendizaje o sistemas de producción simbólica, y los sensores para robots.

**Economía:** En este caso, se ha hecho uso de estos Algoritmos para modelizar procesos de innovación, el desarrollo estrategias de puja, y la aparición de mercados económicos.

**Sistemas inmunes:** A la hora de modelizar varios aspectos de los sistemas inmunes naturales, incluyendo la mutación somática durante la vida de un individuo y el descubrimiento de familias de genes múltiples en tiempo evolutivo, ha resultado útil el empleo de esta técnica.

**Ecología:** En la modelización de fenómenos ecológicos tales como las carreras de armamento biológico, la coevolución de parásito-huesped, la simbiosis, y el flujo de recursos.

**Genética de poblaciones:** En el estudio de preguntas del tipo "¿Bajo qué condiciones será viable evolutivamente un gene para la recombinación?".

**Evolución y aprendizaje:** Los AG se han utilizado en el estudio de las relaciones entre el aprendizaje individual y la evolución de la especie.

**Sistemas sociales:** En el estudio de aspectos evolutivos de los sistemas sociales, tales como la evolución del comportamiento social en colonias de insectos, y la evolución de la cooperación y la comunicación en sistemas multi-agentes.

Aunque esta lista no es, en modo alguno, exhaustiva, sí transmite la idea de la variedad de aplicaciones que tienen los AG. Gracias al éxito en estas y otras áreas, los AG han llegado a ser un campo puntero en la investigación actual.