Biología + Matemáticas + Informática = Algoritmos Genéticos

por

María Teresa Iglesias Otero, Universidade da Coruña, *totero@udc.es*

En los últimos años han adquirido gran importancia los algoritmos que imitan a los mecanismos de la naturaleza para resolver problemas. Algunos de los más importantes son los **autómatas celulares** que simulan la cooperación entre diferentes células, las **redes de neuronas** que simulan el sistema nervioso y los **algoritmos evolutivos** que se inspiran en el proceso biológico de la evolución de las especies.

Los Algoritmos Genéticos son una subclase de algoritmos evolutivos basados en el principio de la supervivencia del más fuerte que en las últimas tres décadas vienen aplicándose con éxito en problemas muy diversos –como el problema de dividir un grupo de n objetos en k categorías o el problema de dibujar, bajo ciertos criterios estéticos, grafos dirigidos– y en muy diferentes ámbitos: economía, medicina, ingeniería o inteligencia artificial.

1.1 Un poco de Historia

Los primeros hechos relacionados con los Algoritmos Genéticos (en adelante AGs) surgieron en 1932 cuando Cannon interpreta la evolución natural como un proceso de aprendizaje muy similar al proceso mediante el cual una persona aprende por ensayo y error. También en 1950 Turing reconoce una conexión entre la evolución

y el aprendizaje de una máquina, pero los primeros intentos serios de relacionar la informática y la evolución surgieron a principios de los años sesenta cuando varios biólogos comenzaron a experimentar con simulaciones de sistemas genéticos; esto es, modelos computacionales que imitan la evolución biológica. El entorno en que viven los animales se representa entonces en el programa por una función que asigna a cada animal su capacidad de llegar a ser adulto y reproducirse.

Los investigadores en computación observaron que esta clase de algoritmos podía utilizarse también para optimizar funciones. De hecho, aunque las primeras descripciones técnicas y definiciones de adaptación provienen de la biología, ya en este contexto, adaptación designa cualquier proceso por el cual una estructura va modificándose de forma progresiva para lograr el comportamiento óptimo en su entorno. Es decir, los procesos adaptativos son básicamente procesos de optimización, pero es difícil aglutinarlos y unificar su estudio porque las estructuras modificables son complejas y su comportamiento es incierto. A menudo, interacciones no aditivas (*epistasis* y no linealidad) hacen imposible determinar el comportamiento de un todo a partir del estudio de sus partes aisladamente. Estas dificultades suscitan un verdadero problema al analista –sin embargo, el proceso adaptativo biológico las maneja de forma rutinaria–.

Una observación cuidadosa de la evolución sufrida por esas estructuras revela generalmente un conjunto básico de modificadores estructurales u operadores, cuyas acciones reiteradas conducen a las modificaciones observadas.

Los AGs están inspirados en la evolución y la mayoría de los organismos evolucionan a través de dos procesos primarios: la selección natural y la reproducción.

Mediante la selección se determina qué miembros de una población sobreviven para reproducirse, y mediante la reproducción se asegura la mezcla y recombinación de los genes de la descendencia.

Esta mezcla del material genético permite que las especies evolucionen mucho más rápidamente de lo que lo harían si tuvieran sólo la copia de los genes de uno de sus progenitores.

El principio fundamental de la selección natural fue formulado por Darwin –anteriormente al descubrimiento de los mecanismos genéticos– al observar que una diferencia entre el crecimiento de una población y los recursos disponibles llevaría a situaciones de competición y adaptación. Desconocedor en aquel momento de los principios básicos de la herencia mendeliana, Darwin conjeturó la fusión de las

cualidades de los padres mezclándose en los organismos de sus descendientes.

La teoría de la evolución defendida por Charles Darwin se sustenta en cuatro argumentos:

* La naturaleza está en constante evolución: las especies cambian continuamente; con el paso del tiempo algunas se extinguen y otras nuevas aparecen.

* El proceso de cambio es gradual y continuo, no se produce a saltos discontinuos o por transformaciones súbitas.

* Los organismos que presentan semejanzas están emparentados.

* El cambio evolutivo es el resultado del proceso de selección natural: los individuos más eficientes a la hora de proporcionarse alimentos y hábitat, y con más aptitudes para dejar un mayor número de descendientes, son los más favorecidos y transmiten a sus hijos los caracteres favorables. En la lucha por la existencia sobreviven los más aptos.

Las cuestiones, relacionadas con los mecanismos hereditarios, a las que la teoría de Darwin no podía responder han sido contestadas con el desarrollo de la Genética: Mendel descubrió los principios básicos de la transferencia de factores hereditarios de padres a hijos. Se estableció que los cromosomas son los principales portadores de la información hereditaria y que los genes, que representan los factores hereditarios, están alineados en cromosomas. El origen de las variaciones en la herencia se explican por la existencia de ciertos cambios (*mutaciones*) en el texto genético, pudiendo sufrir mutación todos los organismos de forma aleatoria. Pero además, en aquellos individuos que se reproducen sexualmente se puede considerar el proceso por el que las características de los organismos se mezclan al combinar su ADN (*cruce*). Este proceso es la fuente de inspiración de los AGs y, en consecuencia, su idea básica es imitar lo que hace la naturaleza. Por esta razón estos algoritmos utilizan un vocabulario tomado de la Genética. Se habla de individuos (genotipos o estructuras) en una población y a menudo estos individuos se denominan cadenas o cromosomas.

El comienzo del desarrollo de los algoritmos genéticos se ha debido realmente al trabajo de John Holland, investigador matemático de la Universidad de Michigan, quien estaba convencido de que era la *recombinación* de grupos de genes, que se realiza mediante el cruce, la parte más importante de la evolución. A mediados de los años 60 desarrolla una técnica de programación, *el Algoritmo Genético*, que se adapta a la evolución tanto por el cruce como por la mutación. Durante la década

siguiente trabajó para ampliar el alcance de este tipo de algoritmos y fruto de ese trabajo publica en 1975 la primera monografía sobre el tema, *Adaptation in Natural and Artifical Systems,* ([4]) en la que se sientan las bases teóricas que fundamentan el desarrollo de los AGs desde el punto de vista computacional, abstrae los conceptos de la genética natural, y los aplica a la economía y al reconocimiento de patrones. Desde entonces el campo de aplicaciones de este tipo de algoritmos no ha dejado de crecer: diseños de turbinas de aviones, predicciones de la evolución bursátil, estudios de progresión de enfermedades (mediante comparación de imágenes tomadas por resonancia magnética), análisis de la estructura del cristal líquido, son algunos ejemplos.

1.2 ¿Por qué utilizar algoritmos genéticos en la optimización?

La razón del creciente interés por los algoritmos genéticos es que éstos son un método global y robusto de búsqueda de las soluciones de problemas. La principal ventaja de estas características es el equilibrio alcanzado entre la eficiencia y eficacia para resolver diferentes y muy complejos problemas de grandes dimensiones.

Lo que aventaja a los AGs frente a otros algoritmos tradicionales de búsqueda es que se diferencian de éstos en los siguientes aspectos:

• Trabajan con una codificación de un conjunto de parámetros, no con los parámetros mismos.

• Trabajan con un conjunto de puntos, no con un único punto y su entorno (su técnica de búsqueda es global.) Utilizan un subconjunto del espacio total, para obtener información sobre el universo de búsqueda, a través de las evaluaciones de la función a optimizar. Esas evaluaciones se emplean de forma eficiente para clasificar los subconjuntos de acuerdo con su idoneidad.

• No necesitan conocimientos específicos sobre el problema a resolver; es decir, no están sujetos a restricciones. Por ejemplo, se pueden aplicar a funciones no continuas, lo cual les abre un amplio campo de aplicaciones que no podrían ser tratadas por los métodos tradicionales.

• Utilizan operadores probabilísticos, en vez de los típicos operadores determinísticos de los técnicas tradicionales.

• Resulta sumamente fácil ejecutarlos en las modernas arquitecturas masivas en paralelo.

• Cuando se usan para problemas de optimización, resultan menos afectados por los máximos locales que las técnicas tradicionales (i.e., son métodos robustos).

1.3 ¿Cómo funcionan?

Para modelar computacionalmente los sistemas adaptativos naturales mediante algoritmos genéticos es necesario extraer y generalizar los factores fundamentales de los procesos biológicos.

En el contexto de la optimización funcional, que un proceso evolutivo actúe sobre una población de individuos se corresponde con la búsqueda del óptimo en el conjunto de las soluciones potenciales. (Generalmente lo algoritmos genéticos se formulan para problemas de maximización, lo que, obviamente, no supone ninguna restricción.)

Dado que un AG actúa como un método de búsqueda multidireccional sobre una población de posibles soluciones, ésta sufre la simulación de una evolución: en cada generación las soluciones relativamente buenas se reproducen, mientras las relativamente malas mueren. Para distinguir las diferentes soluciones se usa una función evaluadora que juega el papel del entorno. Esta función se denomina *función de ajuste* o función de idoneidad y, en el campo de la optimización funcional, es la función objetivo del problema en cuestión; es decir, la función a optimizar.

El funcionamiento de un algoritmo genético clásico comienza generando una *población inicial* aleatoria de candidatos a ser el óptimo y usa la información contenida en ella con el fin de producir iterativamente nuevas y "mejores" poblaciones. Esa "calidad" se entiende en términos de la medida que de su idoneidad proporciona la función de ajuste.

Para ello, en primer lugar –y una vez codificados los miembros de la poblacion inicial–, se evalua su idoneidad y se seleccionan los mejores cromosomas. Algunos de estos individuos seleccionados sufren ciertas alteraciones por la acción de diferentes *operadores genéticos* encargados de introducir nuevos elementos en la población. En los algoritmos genéticos clásicos, dos son los operadores genéticos a destacar: el *cruce* y la *mutación*.

El cruce combina los genes de dos cromosomas padres para formar dos descendientes mezclando segmentos de los dos padres. Por ejemplo, si los padres están representados por vectores 5 dimensionales P_1=(a_1,b_1,c_1,d_1,e_1) y P_2=(a_2,b_2,c_2,d_2,e_2) el cruce de cromosomas después del segundo gen produce los hijos H_1=(a_2,b_2,c_1,d_1,e_1)

y H_2=(a_1,b_1,c_2,d_2,e_2).

La mutación por el contrario es un operador unario, que arbitrariamente altera uno o más genes de un cromosoma:

$$(1, \underline{0}, 0, 0, 0, 1, 1) \rightarrow (1, \underline{1}, 0, 0, 0, 1, 1).$$

Los descendientes obtenidos a través de la *selección* y la *recombinación*, constituyen lo que se conoce como una *generación*. El proceso se repite durante generaciones hasta que se alcanza una cierta condición de parada. Es decir, el algoritmo finaliza cuando se ha ejecutado un número determinado de iteraciones prefijado de antemano, cuando se ha encontrado el óptimo (i.e. cuando, después de un cierto número de generaciones, el programa ha convergido al mejor individuo, que representa la mejor solución al problema) o bien cuando se ha obtenido en la población un nivel de idoneidad medio superior a un cierto nivel de control (*valor umbral*).

Existen pues una serie de puntos comunes a todos los algoritmos genéticos que los caracterizan:

• *La existencia de una representación genética de las posibles soluciones del problema:* Los AGs tradicionales generalmente emplean, como individuos de la población o cromosomas, vectores (*cadenas*) de longitud fija. Cada una de las posiciones de la cadena se conoce como un *gen*. Cada carácter de un individuo puede tener diferentes manifestaciones, es decir el gen puede tomar diferentes valores llamados *alelos* (v.g. el color del pelo y sus posibles variantes.) Cada uno de estos alelos aparece en una posición determinada de la cadena (*locus*). Cada individuo representará una solución potencial del problema.

• *La creación de una población inicial de posibles soluciones*.

• *El uso de una función evaluadora* que desempeña el papel del medio ambiente, y que es la encargada de medir la "bondad" de las soluciones en términos de su adaptación al medio y que, además, permite seleccionar a los mejores individuos.

• *El empleo de operadores genéticos*, que alteran la composición de los hijos durante la reproducción, mediante cruces, mutaciones o inversiones, por ejemplo.

• *La existencia de diversos parámetros* como son el tamaño de la población o la probabilidad de aplicar los operadores genéticos. En el caso particular de la mutación se asignan valores de probabilidad muy pequeños ($p_m = 0.01$ o $p_m = 0.001$) para mantener la idea biológica de que la mutación es un hecho raro en la naturaleza.

La elección de la codificación de las posibles soluciones, mediante cadenas (en

cuanto a la longitud de las mismas o los valores de los alelos) es lo que se conoce como la cuestión de la *representación*, y es fundamental en la eficacia y/o eficiencia del algoritmo.

La elección de la codificación de los datos debe cumplir dos objetivos: ser apropiada para las técnicas de optimización y reflejar, en la medida de lo posible, la estructura del problema.

Por su sencillez, la codificación basada en el alfabeto binario (*codificación binaria*) es la más utilizada, aunque para ejemplos concretos se utilizan otro tipo de codificaciones con alelos naturales, en árboles, etc, dependiendo de la estructura del problema a tratar. En particular, si se pretende codificar una estructura que encierra una información dos dimensional –pensemos por ejemplo en una matriz

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} -$$

se podría representar fácilmente por una cadena de doce genes, concatenando sus filas (o sus columnas):

$$(a_{11}, a_{12}, a_{13}, a_{14}, a_{21}, a_{22}, a_{23}, a_{24}, a_{31}, a_{32}, a_{33}, a_{34}, a_{41}, a_{42}, a_{43}, a_{44}),$$

sin embargo, esta codificación ignora el hecho de que a_{11} y a_{21} eran adyacentes en la matriz; por contra, a_{24} y a_{31} lo son ahora, cuando antes estaban distantes.

La codificación de imágenes es otro ejemplo en el que se pone de manifiesto la importancia de establecer una representación apropiada de los datos, que no pierda la correlación existente entre los bits adyacentes; es decir, que no destruya información importante para el problema.

Un interesante ejemplo que muestra que el diseño de un AG no es una cuestión trivial es el conocido *problema del viajante* o TSP (Traveling salesman problem): "*Un viajante debe recorrer un número determinado de ciudades pasando por todas y cada una de ellas una sola vez, y debe hacerlo encontrando el recorrido óptimo*".

Por ser un *NP-hard problem*, la única forma en la que este problema había sido tratado es por métodos heurísticos. Sin embargo, este tipo de métodos no proporcionan ninguna información sobre la calidad de la solución obtenida. Además, cuando el número de ciudades a recorrer es grande (por ejemplo 100) entonces las

dimensiones del problema aumentan considerablemente[1]) y los métodos no son operativos desde el punto de vista práctico.

Al abordar el TSP usando AGs, se ha observado que la codificación binaria no es la más apropiada. Entre las distintas alternativas propuestas la más natural es aquella en la que se codifica una ruta por un vector compuesto de tantos elementos como ciudades se deban recorrer, siendo el primer elemento del vector la primera ciudad y el último elemento la última ciudad en ser visitada (*representación del camino* o "path representation" [5].)

Por supuesto, el tipo de representación elegida condiciona el tipo de operadores genéticos que actúan sobre ella.

Como ya se ha indicado, el operador de cruce clásico consiste en tomar 2 cadenas padres, seleccionar un punto de cruce entre dos alelos de las mismas, e intercambiar cabezas y colas:

Padres	Hijos
0110|01010111	0110|10111000
1011|10111000	1011|01010111

En la representación del camino para el TSP este cruce podría generar recorridos que no cumplan los requisitos exigidos:

Padres	Hijos
$AB\|CDEF$	$ABBFAE$
$CD\|BFAE$	$CDCDEF$

Se ha comprobado que uno de los operadores de cruce más eficaces es el *operador de recombinación de bordes* ("edge recombination operator") que actúa del siguiente modo:

Dados los padres $ABCDEFGHI$ y $DABHGFICE$, se construye una tabla en la

[1]En ese caso el número total de diferentes rutas posibles es $100! \simeq 10^{258}$. El número de átomos del universo es aproximadamente 10^{128}.

que se reflejan las ciudades adyacentes a cada una de las del recorrido completo:

A	B, I, D
B	A, C, H
C	B, D, I, E
D	C, E, A
E	D, F, C
F	E, G, I
G	F, H
H	G, I, B
I	A, H, F, C

La construcción del hijo comienza seleccionado una población inicial de uno de los dos padres. Se selecciona aquella que tiene menos ciudades adyacentes a ella (en caso de empate se escoge una aleatoriamente.) Por ejemplo supongamos elegida la ciudad A. Esta ciudad está conectada con las ciudades B, I y D. La siguiente ciudad que formará parte del descendiente saldrá de una de esas tres. B y D tienen 3 conexiones a otras tantas ciudades, mientras que I tiene 4. Se efectua entonces una elección aleatoria entre B y D. Elijamos la ciudad D. De nuevo las candidatas para ser la siguiente ciudad en el recorrido son las ciudades directamente conectadas a D. Se escoge E que tiene 3 conexiones frente a las 4 de C. Continuando el proceso se obtiene finalmente el descendiente:

$$ADEFGHBCI$$

Se ha probado este operador en problemas con 30, 50 y 75 ciudades y se ha comprobado que proporciona una solución mejor que la mejor secuencia conocida (ver [11] para más detalles).

Pero, aparte del problema de la representación y del diseño del método de cruce o mutación, queda la cuestión mas importante:

1.4 ¿Por qué funcionan estos algoritmos?

Uno de los argumentos básicos de la teoría de la evolución es que los individuos que presentan semejanzas están emparentados. Basándose en este principio, Holland observó que ciertos conjuntos de individuos con determinadas similitudes en algunas posiciones de sus cadenas poseían propiedades comunes buenas y otros por contra eran peores.

Por ejemplo en la función $f(x) = x^2$, si codificamos en cadenas de cuatro bits los que empiezan con un 1 tienen imágenes de valor mayor. Es decir, el valor medio sobre las cadenas de la forma 1*** está por encima de la media global de la población. Abstrayendo esta idea Holland define el concepto de *esquema* (H) en una codificación binaria con cadenas de longitud ℓ como

$$H = h_{\ell-1} \dots h_0 \in \{0, 1, *\}^\ell \longleftrightarrow H = \{s_{\ell-1} \dots s_0 \,/\, h_j \neq * \rightarrow s_j = h_j\}\,.$$

Es decir, un esquema representa un cierto subconjunto de la población $\{0,1\}^\ell$ en el que sus individuos se diferencian, a lo sumo, en las posiciones de los asteriscos. Por ejemplo, el esquema $H = 10 * 01*$ se corresponde con el conjunto de cadenas

$$\{\underline{10}0\underline{10}, \underline{10}0\underline{11}, \underline{10}1\underline{01}0, \underline{10}1\underline{01}1\}\,.$$

Por otra parte, cualquier conjunto C de cadenas define un esquema, basta considerar la proyección sobre la j-ésima coordenada

$$\begin{array}{rccc} \pi_j: & \{0,1\}^\ell & \rightarrow & \{0,1\} \\ & s_{\ell-1} \dots s_0 & \mapsto & s_j \end{array}$$

y definir el esquema H como:

$$h_j = \begin{cases} 0 & si\ \pi_j(H) = \{0\} \\ 1 & si\ \pi_j(H) = \{1\} \\ * & si\ \pi_j(H) = \{0,1\}\,. \end{cases}$$

De hecho, el conjunto de cadenas que se pueden generar por cruces de elementos del conjunto C es exactamente H. Por ejemplo, si $C = \{001011, 011111\}$ entonces cada cadena de $H = 0 * 1 * 11$ se puede generar cruzando los elementos de C, incluso 011011 y 001111, que no estaban inicialmente en C.

Obviamente en unos esquemas sus elementos guardan más parecido entre sí que en otros. Para cuantificar esta idea existen dos conceptos: el *orden* de un esquema que es el número de alelos fijos en el esquema y la *longitud de definición* que es la distancia entre el primero y el último alelo fijos. Por ejemplo, si $H = 00 * *1*$, entonces $o(H) = 3$ y $\delta(H) = 4$.

Claramente, a mayor longitud de definición (o a mayor orden) del esquema, más probabilidad de sufrir un cruce o una mutación que destruya la estructura del mismo. Parece pues que los esquemas cortos, de orden bajo son más proclibes a mantenerse en la población y si estos esquemas tienen representantes en la población con

idoneidad alta, éstas cadenas serán seleccionadas con mayor frecuencia y generarán más representantes en la generación siguiente. Esto es grosso modo lo que dice el *Teorema de los Esquemas* que Holland demostró en 1975 y que es el resultado teórico más importante que existe en este campo. En esencia, este teorema afirma que el algoritmo dirige la búsqueda del óptimo a través de ciertos subconjuntos de la población; es decir, explora el espacio de búsqueda a través aquellas áreas que, en media, son más adecuadas.

Examinemos con un poco de detalle el efecto de la reproducción sobre el número de representantes de un esquema H. Dado que durante la reproducción un cromosoma es seleccionado con una probabilidad proporcional a su idoneidad

$$\frac{f(s)}{\sum\limits_{s\in P(t)} f(s)},$$

donde $P(t)$ denota la población en la generación t-ésima, entonces si partimos de una población de N elementos, el número esperado de representantes de H en la iteración siguiente es:

$$\mathbf{E}(n(H,t+1)) = n(H,t)\cdot N\cdot \frac{f(H,t)}{\sum\limits_{s\in P(t)} f(s)}$$

donde $\mathbf{E}$ denota el operador esperanza, $n(H,t)$ es el número de cadenas del esquema en la generación t-ésima y

$$f(H,t) = \frac{\sum_{s\in H} f(s)}{n(H,t)}$$

es el valor medio del esquema en esa generación.

Si ahora consideramos la acción de los operadores de cruce y mutación, la anterior ecuación se transforma en:

$$\mathbf{E}(n(H,t+1)) = n(H,t)\cdot \frac{f(H,t)}{\overline{f}}\,[1-\alpha(H)]$$

donde $\overline{f} = \frac{\sum_{s\in P(t)} f(s)}{N}$ es la idoneidad media de la población y $\alpha(H)$ depende de la estructura de H y de las probabilidades de cruce y mutación, p_c y p_m, pero no de t, pues

$$\alpha(H) = p_c\cdot \frac{\delta(H)}{\ell-1}\cdot (1-p_m)^{o(H)}.$$

Esta expresión, al despreciar los términos de grado ≥ 2 en el binomio de Newton, y como consecuencia de que $p_m \simeq 0.01$, se convierte en la formulación definitiva del *Teorema Fundamental de los algoritmos genéticos (o Teorema de los esquemas):*

$$\mathbf{E}(n(H,t+1)) \geq n(H,t) \cdot \frac{f(H,t)}{\overline{f}} \left[1 - p_c \cdot \frac{\delta(H)}{\ell - 1} - o(H) \cdot p_m\right].$$

De modo que que si H es un esquema con una idoneidad superior a la media de la población, se espera que se incremente el número de cadenas con la estructura de H en la generación siguiente, siempre que α sea pequeña. Es decir, la tesis de este teorema se puede interpretar diciendo que *"esquemas cortos, de orden bajo, con idoneidad sobre la media, incrementan el número de representantes en generaciones sucesivas"*. Este tipo de esquemas –que parecen jugar un papel importante en la actuación de los AGs–, se conocen como bloques de construcción o *building blocks*.

Parece pues que yuxtaponiendo bloques sólidos de tamaño pequeño se pueden llegar a construir individuos cada vez mejores, al igual que los niños con piezas pequeñas construyen estructuras fuertes y sólidas en sus mecanos. Esto llevó a pensar durante algún tiempo que funciones que se pudiesen definir utilizando esquemas cortos, de orden bajo y altamente idoneos serían fáciles de optimizar por los AGs. Esta afirmación –conocida como *la hipótesis de los bloques constructivos*– parece muy razonable; de hecho, se han diseñado AGs para aplicaciones variadas que son una evidencia empírica de que, para diferentes tipos de problemas, dicha hipótesis es correcta. Sin embargo, en 1993 Forrest y Mitchell [2] definen dos funciones (*Royal Road functions*) que la contradicen. La primera de ellas, $\Re_1$, se define a partir de los esquemas:

$$\begin{aligned}
H_1 &= 1^{(8)} *^{(56)} \\
H_2 &= *^{(8)} 1^{(8)} *^{(48)} \\
H_3 &= *^{(16)} 1^{(8)} *^{(40)} \\
H_4 &= *^{(24)} 1^{(8)} *^{(32)} \\
H_5 &= *^{(32)} 1^{(8)} *^{(24)} \\
H_6 &= *^{(40)} 1^{(8)} *^{(16)} \\
H_7 &= *^{(48)} 1^{(8)} *^{(8)} \\
H_8 &= *^{(56)} 1^{(8)}
\end{aligned}$$

donde $\alpha^{(n)}$ denota la cadena $\overbrace{\alpha\alpha\ldots\alpha}^{n}$ y $\Re_1 : \{0,1\}^{64} \longrightarrow \mathbb{R}$, viene dada por $\Re_1(s) = \sum_{i:s\in H_i} 8$, para cada $s \in \{0,1\}^{64}$. Por ejemplo, $\Re_1(1^{(8)}0^{(56)}) = 8$

mientras que $\Re_1(1^{(16)}0^{(48)}) = 16$. El máximo de esta función se alcanza en la cadena $s = 11\ldots11 = 1^{(64)}$.

La segunda función Royal Road, $\Re_2$, se define de forma similar a la primera, a partir de los esquemas anteriores añadiendo los siguientes:

$$\begin{aligned} H_9 &= 1^{(16)}\,*^{(48)} \\ H_{10} &= *^{(16)}1^{(16)}\,*^{(32)} \\ H_{11} &= *^{(32)}1^{(16)}\,*^{(16)} \\ H_{12} &= *^{(48)}1^{(16)} \\ H_{13} &= 1^{(32)}\,*^{(32)} \\ H_{14} &= *^{(32)}1^{(32)} \end{aligned}$$

y con $\Re_2(s) = \sum_{i:s\in H_i} c_i$, donde, como antes, $c_i = 8$ para $1 \leq i \leq 8$ y $c_i = 16$ (resp. $c_i = 32$) para $9 \leq i \leq 12$ (resp. $13 \leq i \leq 14$.) Obviamente, esta función alcanza el máximo en la misma cadena que la anterior función.

Es razonable esperar que estas dos funciones sean fáciles para los AGs, porque están definidas por medio de esquemas cortos, de orden bajo, que se pueden utilizar como un camino directo (*Royal Road*) hacia la solución óptima[2].

Pero, además, la función $\Re_2$ debería ser más fácil que $\Re_1$ ya que se utilizan más esquemas en la definición.

Como medida de la dificultad, que el algoritmo encuentra para optimizar funciones, se puede considerar el número medio de generaciones necesarias para que se cumplan las tres condiciones siguientes:

- al menos un elemento de la población tenga valor máximo,

- la idoneidad media de la población sea superior al 90% del valor máximo[3],

- la desviación típica tenga un valor menor o igual que el 5% de la media.

Se ha demostrado que estas funciones son difíciles para los AGs y de hecho, comparando el comportamiento de ambas, resulta que $\Re_2$ es realmente más difícil que $\Re_1$. Esto no contradice el teorema de los esquemas puesto que la explicación

[2]Se espera que el algoritmo combine el esquema H_1 con el esquema H_2, los esquemas H_3 y H_4, etc., para llegar al óptimo de forma rápida.

[3]En los casos en los que no se conoce el valor del máximo, se exige que un porcentaje alto de la población alcance un valor umbral prefijado.

del comportamiento de los algoritmos genéticos mediante el citado teorema es incompleta; no precisa cómo se comporta el algoritmo frente a esquemas sin representantes en la población ("todavía") y, en consecuencia, no informa de la dirección de búsqueda en las iteraciones sucesivas. De hecho, las medias $f(H,t)$ son medias de esquemas presentes en la población actual y, por tanto, el algoritmo carece de referencias a nuevos esquemas que se introduzcan como consecuencia de cruces o mutaciones. Tampoco se informa sobre lo que sucede si no hay "buenos" esquemas en la población de partida y se debe esperar a que eventuales cruces o mutaciones los generen. Mientras ello sucede el algoritmo carece de información que le guie a zonas donde estén las cadenas mejores. Eso puede hacer que emplee demasiado tiempo en la búsqueda del óptimo y que no resulte eficiente.

Llegados a este punto, parece razonable preguntarse qué funciones valdrá la pena optimizar aplicándoles un algoritmo genético. De hecho, las técnicas clásicas de optimización están diseñadas para ciertos conjuntos de funciones (lineales, diferenciables...) y este conocimiento se utiliza explícitamente en el algoritmo del método, mientras que en el caso de los AGs no está claro qué suposiciones se deben hacer sobre la función a la que aplicarle tal algoritmo. Es decir,

1.5 ¿Cuáles son las propiedades que identifican a las funciones difíciles de optimizar por los algoritmos genéticos?

Algunas conclusiones derivadas erróneamente del teorema de los esquemas condujeron a la definición de propiedades que se suponía que caracterizaban a las funciones difíciles: "serán difíciles aquellas funciones para las cuales los bloques constructivos engañen al algoritmo dirigiéndolo hacia esquemas con media peor, es decir, a subóptimos". Son las *funciones decepcionantes*. Para introducir formalmente la decepción es necesario considerar algunos conceptos previamente.

Se dice que dos esquemas son *competitivos* si los bits definidos de ambos están en las mismas posiciones, pero al menos uno de los valores de esos bits es diferente. En ese caso, el conjunto de todos los esquemas competitivos entre sí define una partición en el espacio de búsqueda. Por ejemplo, los esquemas

$$\begin{aligned} &0*0*...* \\ &0*1*...* \\ &1*0*...* \\ &1*1*...* \end{aligned} \tag{1.1}$$

son competitivos y definen claramente una partición.

El *orden de la partición* es el orden de cualquiera de sus esquemas. Se dice que una partición P *subsume* a una partición P' si cada esquema de P' es un subconjunto de un esquema de P. Por ejemplo, la partición $P = \{**0, **1\}$ subsume a la partición $P' = \{*00, *01, *10, *11\}$.

Pues bien, una función *contiene decepción*, ([12]), si existen dos particiones P y P', P' de orden mayor que P y subsumida por P, tales que el esquema *ganador*[4] de P' tiene al menos un bit, entre las posiciones de bits definidos y comunes a P y P', cuyo valor difiere del valor del bit correspondiente en el ganador de P.

El ejemplo más sencillo de decepción –conocido como el *problema decepcionante mínimo* o MDP– se debe a Goldberg ([3]) y se puede resumir como sigue:

Supongamos una partición de esquemas competitivos de orden dos como los dados en (1.1) y supongamos que, para una cierta función f las idoneidades medias de esos esquemas son $f(00), f(01), f(10)$ y $f(11)$ siendo $f(11)$ el óptimo global. Ahora se introduce el elemento de decepción: uno (o ambos) de los esquemas subóptimos de orden 1 son mejores que el óptimo de orden 1; esto es, una (o ambas) de las condiciones siguientes se verifican

$$\begin{aligned} f(0*) &> f(1*) \\ f(*0) &> f(*1), \end{aligned}$$

de las que se deduce que

$$\begin{aligned} f(00) + f(01) &> f(10) + f(11) \\ f(00) + f(10) &> f(01) + f(11). \end{aligned}$$

Dado que estas inecuaciones son incompatibles, tomemos una de ellas, por ejemplo la primera. Esa inecuación, junto con el hecho de que $f(11)$ es el óptimo conducen a una clasificación del problema en dos tipos distintos

$$\begin{aligned} Tipo\,\mathrm{I} &: f(01) > f(00) \\ Tipo\,\mathrm{II} &: f(00) \geq f(01). \end{aligned}$$

Ambos casos contienen decepción y no se puede expresar ninguno de ellos como combinación lineal de los valores individuales de los alelos[5].

[4] Por *ganador* se entiende el esquema de mayor idoneidad entre todos los de la partición.
[5] En términos biológicos estamos ante un problema epistático.

A pesar de que existe decepción en los dos casos, el comportamiento de la función es diferente en cuanto a la convergencia en un tipo o en otro. En el tipo I la decepción puede inicialmente dirigir al algoritmo lejos del óptimo global pero, después de un cierto número de generaciones, el AG es capaz de encontrar el máximo; es decir, el tipo I no es difícil para el algoritmo. Esto no sucede necesariamente en el MDP del tipo II. De hecho, si en la población inicial hay el mismo número de representantes de cada uno de los cuatro esquemas, entonces la situación es similar a la del tipo I (tras un engaño inicial el algoritmo converge a la mejor solución.) Sin embargo, si el número de copias de $0 * 0 * \cdots *$ es superior al de los otros tres esquemas, el algoritmo sufre un engaño y converge a una solución subóptima.

Este ejemplo, junto con las funciones Royal Road –que no contienen decepción– demuestra que la decepción no es necesaria ni suficiente para que un problema sea difícil de optimizar mediante un algoritmo genético.

La caracterización de las funciones difíciles de optimizar es una cuestión abierta todavía, sin embargo, en 1991, Rawlins ([8]) propone como medida de la dificultad un estadístico, *epistasis*, inspirado en un concepto biológico del que hereda el nombre y que hace referencia a la relación de dependencia entre diversos genes de un cromosoma[6].

1.6 Epistasis y dificultad

Rawlins observó que en ocasiones un gen, o un grupo de genes, enmascara el efecto de otro (fenómeno que en biología se conoce como epistasis) y sugirió que en el contexto de los AGs se observan dos extremos en el comportamiento de las funciones a optimizar:

- Epistasis cero. En este caso cada gen es independiente de cualquier otro gen. [...] Está claro que esta situación puede ocurrir sólamente cuando la función objetivo se puede expresar como una combinación lineal de funciones, que dependen, cada una de ellas de un gen.

- Epistasis máxima. En este caso ningún subconjunto de genes es independiente de cualquier otro gen. Cada gen depende de los demás para su idoneidad.[...]. Esta situación es equivalente a que la función objetivo sea una función aleatoria.

[6]La razón de la dificultad de las funciones Royal Road es que en el valor de una cadena tan importante como la cantidad de unos es la colocación de los mismos. Obsérvese que, por ejemplo, $\Re_2(11\ldots1) = 192$, $\Re_2(11\ldots10) = 136$ y $\Re_2(01\ldots10) = 80$.

La formalización de esta idea se debe a Davidor ([1]) que en 1991 define la bondad de un alelo, a, del alfabeto binario, situado en la posición i-ésima de una cadena de una población P de individuos, como

$$f(a,i) = \frac{1}{N_i(a)} \sum_{s \in P_{i,a}} f(s).$$

Aquí la suma se efectúa sobre el conjunto de todas las cadenas de la población que tienen el alelo a en la posición i, y $N_i(a)$ es el cardinal de $P_{i,a}$. Asimismo, define el valor de exceso del alelo como la diferencia entre el valor anterior y la media de la población, $\overline{f}$, i.e., $E_i(a) = f(a,i) - \overline{f}$. Además, para una cadena s, el valor de exceso génico es la suma de los valores de exceso de sus alelos

$$\widetilde{E}(s) = \sum_{i=0}^{\ell-1} E_i(s_i)$$

y su valor génico previsto es $\widetilde{f}(s) = \widetilde{E}(s) + \overline{f}$. Finalmente, la epistasis de una cadena s es la diferencia entre el valor de la cadena y su valor previsto; es decir,

$$\varepsilon(s) = f(s) - \widetilde{f}(s).$$

Reuniendo las fórmulas anteriores, Van Hove ([10]), en 1994, reescribe la definición de Davidor y define la epistasis de una cadena $s \in P = \{0,1\}^{\ell}$ como

$$\varepsilon(s) = f(s) - \sum_{i=0}^{\ell-1} \frac{1}{2^{\ell-1}} \sum_{t \in P_{i,s_i}} f(t) + \frac{\ell-1}{2^{\ell}} \sum_{t \in P} f(t).$$

La anterior definición es posible reescribirla en forma matricial. Para ello, basta definir los vectores $\overrightarrow{\varepsilon}$ y $\overrightarrow{\mathbf{f}}$:

$$\overrightarrow{\varepsilon} = \begin{pmatrix} \varepsilon(00...00) \\ \varepsilon(00...01) \\ \vdots \\ \varepsilon(11...11) \end{pmatrix}, \qquad \overrightarrow{\mathbf{f}} = \begin{pmatrix} f(00...00) \\ f(00...01) \\ \vdots \\ f(11...11) \end{pmatrix}$$

que caracterizan a la epistasis y a la función objetivo, respectivamente. Estos vectores están relacionados a través de la matriz cuadrada 2^{ℓ}-dimensional $\mathbf{E}_{\ell} = (e_{ij})$ con

$$e_{ij} = \frac{1}{2^{\ell}}(\ell + 1 - 2d_{ij}) \quad 0 \leq i,j \leq 2^{\ell} - 1,$$

donde d_{ij} es la distancia Hamming entre i y j es decir, el número de bits en los cuales las representaciones binarias de i y j difieren[7]. Entonces,

$$\overrightarrow{\varepsilon} = \overrightarrow{\mathbf{f}} - \mathbf{E}_\ell \overrightarrow{\mathbf{f}}$$

y la *epistasis global* de f se define como la norma del vector $\overrightarrow{\varepsilon}$,

$$\varepsilon_\ell(f) := \sqrt{\sum_{s \in P} \varepsilon^2(s)} = \| \overrightarrow{\varepsilon} \| .$$

Volviendo a la idea que originó el concepto de epistasis, parece claro que f y rf deben tener las mismas relaciones de dependencia entre los bits, para cualquier número real r, lo que exige una normalización del concepto. Así, la *epistasis normalizada* de una función f viene dada por:

$$\varepsilon^*_\ell(f) \;=\; \varepsilon^2_\ell \left(\frac{f}{\| \overrightarrow{\mathbf{f}} \|} \right) = \frac{{\varepsilon_\ell}^2(f)}{\| \overrightarrow{\mathbf{f}} \|^2} = \frac{{}^t \overrightarrow{\mathbf{f}} (\mathbf{I}_\ell \text{-} \mathbf{E}_\ell) \overrightarrow{\mathbf{f}}}{{}^t \overrightarrow{\mathbf{f}} \overrightarrow{\mathbf{f}}} = \cos^2 (\overrightarrow{\mathbf{f}}, \mathbf{F}_\ell \overrightarrow{\mathbf{f}}),$$

donde $\mathbf{F}_\ell = \mathbf{I}_\ell \text{-} \mathbf{E}_\ell$ es una proyección ortogonal (es idempotente y simétrica); de lo que se sigue que $0 \leq \varepsilon^*_\ell(f) \leq 1$.

En cuanto a los valores extremos de la epistasis normalizada, se ha comprobado ([9]) que una función tiene epistasis nula si y sólo si se escribe como combinación lineal de funciones cada una de las cuales depende sólo de un bit, recuperando así la idea intuitiva de Rawlins. Por otra parte, el valor máximo real que se alcanza para funciones no negativas es $1 - \frac{1}{2^{\ell-1}}$ y lo alcanzan las funciones de ajuste que son nulas excepto para dos puntos con distancia Hamming máxima entre ellos y con el mismo valor en dichos puntos[8] (*camel functions*).

Aunque a primera vista parezca extraño que esas funciones sean más difíciles de optimizar que la función de Dirac –en donde la búsqueda del extremo se convierte prácticamente en una búsqueda aleatoria– sin embargo, se debe tener en cuenta que el máximo de la función de Dirac es "más estable" que los dos máximos m_1, m_2 ($m_1 = m_2$) de estas funciones, en el sentido siguiente: una vez que se halla el máximo de la función de Dirac, el AG continuará seleccionándolo con una probabilidad alta, debido a su valor no nulo. Si bien es cierto que, combinado con otros puntos –por ejemplo por medio de un cruce– se perderá sin embargo, el cruce casi siempre

[7] Por ejemplo, $d(1101, 1100) = 1$ y $d(1101, 1010) = 3$.

[8] Por ejemplo $f(0 \ldots 0) = f(1 \ldots 1) = 1$ y $f(s) = 0$ para todo $s \neq 0, 2^{\ell-1}$

utilizará dos copias de ese máximo (precisamente por su alta idoneidad) y esto no sólo le permitirá sobrevivir, sino que incluso aumentará su proporción en la población.

En el caso de las funciones camello las cosas son algo distintas ya que m_1 y m_2 tienen igual probabilidad de ser seleccionadas desde el momento en que se encuentren en la población. Esta probabilidad, que por otra parte es alta, hará que el cruce entre ambos los destruya dirigiendo al algoritmo lejos de ellos. Desde luego, en las situaciones prácticas, esto es, cuando la longitud de las cadenas es relativamente grande, ambas funciones tienen prácticamente la misma epistasis normalizada (aproximadamente igual a 1) y el resultado anterior es de interés principalmente teórico.

La siguiente tabla ([7]) muestra la correlación existente entre epistasis y dificultad para las funciones lineales, las funciones camello y las funciones Royal Road. Los resultados han sido obtenidos experimentalmente trabajando con cadenas binarias de longitud $\ell = 64$, con una población de tamaño 128 y con probabilidades de cruce y de mutación $p_c = 0.7$ y $p_m = 0.002$, respectivamente. Como medida de la dificultad se considera el número medio de generaciones ($G(f)$) –promediado sobre 30 ejecuciones del algoritmo– necesarias para que se verifiquen las tres condiciones dadas en la página 13:

f	$G(f)$	$\varepsilon^*(f)$
$lineal$	37.4	0.00
$\Re_1$	$\gg 1500$	0.93916
$\Re_2$	$\gg 1500$	0.93983
$camel$	$\gg 1500$	0.99999

Aunque la epistasis por sí sola no es, en general, suficiente para predecir la dificultad, se ha comprobado que la epistasis es la que explica el diferente comportamiento de los tipos I y II frente a la convergencia en el MDP [6]). De hecho, se ha demostrado que para una función como la del MDP la epistasis sólo puede tomar tres valores diferentes, y en el caso de epistasis más baja no puede haber decepción. Por otra parte, siempre que existe decepción entonces la epistasis diferencia los tipos I y II, ocurriendo el tipo II cuando el valor de la epistasis es el más alto de los tres posibles y el tipo I para el valor intermedio. Es decir, aunque decepción y epistasis son esencialmente independientes se refuerzan mutuamente.

Bibliografía

[1] Y. Davidor, *Epistasis Variance: A viewpoint on GA-hardness*, Found. Gen. Algorithms 1, ed. G.J. E. Rawlins, Morgan Kaufmann Publ., 1991.
[2] S. Forrest & M. Mitchell, *Relative building block fitness and the building block hypothesis*, Found. Gen. Algorithms 2, ed. L.D. Whitley, Morgan Kaufmann Publ., 109-126, 1993.
[3] D. E. Goldberg, Genetic Algorithms in Search, Optim. Mach. Learning, Addison-Wesley, 1989.
[4] J.H. Holland, Adaptation in Natural and Artifical Systems, Ann Arbor, 1975.
[5] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer, 1996.
[6] B. Naudts, D. Suys & A. Verschoren, *Epistasis, deceptivity and Walsh transforms*, Bull. Soc. Math. Belg. Simon Stevin, 6, 147-154, 1999.
[7] B. Naudts, D. Suys, & A. Verschoren, *Generalized Royal Road Functions and their Epistasis*, Comp. and Art. Intelligence, 19, 317-334, 2000.
[8] G.J.E. Rawlins, Foundations of Genetic Algorithms, Morgan Kaufmann Publ., 1991.
[9] D. Suys & A. Verschoren, *Extreme Epistasis*, ITHURS'96, II, 251-258, León, 1996.
[10] H. Van Hove & A. Verschoren, *On Epistasis*, Comp. and Art. Intelligence 14(3), 271-277, 1994.
[11] L. D. Whitley, T. Starkweather & D'A. Fuquay, *Scheduling Problems and Travelling Salesman: The Genetic Edge Recombination Operator*, Proc. Third Int. Conf. Genetic Algorithms, 133-140, ed. Morgan Kaufmann Publ., 1989.
[12] L. D. Whitley, *Fundamental Principles of Deception in Genetic Search,*, Found. Gen. Algorithms 1, 221-241, ed. G.J.E. Rawlins, Morgan Kaufmann Publ., 1991.