Aprendizaje Automático: Algoritmos genéticos.

Dr. Alejandro Guerra Hernández
Universidad Veracruzana
Facultad de Física e Inteligencia Artificial
Maestría en Inteligencia Artificial
Sebastián Camacho No. 5, Xalapa, Ver., México 91000
aguerra@uv.mx
www.uv.mx/aguerra

Abril 1, 2004

Los algoritmos genéticos proveen un método de aprendizaje inspirado vagamente en la simulación de un proceso de evolución. Las hipótesis son casi siempre descritas como cadenas de bits, cuya interpretación depende de la aplicación del algoritmo. Las interpretaciones incluyen pares atributo-valor, expresiones simbólicas, y aún programas de cómputo. La búsqueda de la hipótesis apropiada comienza con una población inicial, o colección, de hipótesis posibles. Los miembros de una población dan lugar a su descendencia por medio de operaciones como *cruce* y *mutación*, etiquetados así por el simil con la evolución biológica. En cada generación, los elementos de una población son evaluados con respecto a una función de adaptación (*fitness*), determinando así aquellos elementos que tienen una mayor probabilidad de reproducirse en la siguiente generación. En esta sesión presentaremos los algoritmos genéticos donde las hipótesis se representan como cadenas de bits, y una técnica de programación genética, en donde las hipótesis se describen como programas de cómputo.

Observen que, en lugar de llevar a cabo una búsqueda de lo general a lo específico en el espacio de hipótesis (find-S y eliminación de candidatos), o de lo simple a lo complejo (ID3, C4.5), un algoritmo genético genera una serie de hipótesis sucesoras por repetición de mutaciones y combinaciones de las mejores hipótesis conocidas hasta ese momento. En cada iteración del algoritmo, una colección de hipótesis conocida como la *población actual*, se altera remplazando una fracción de ella, por descendientes de las hipótesis que en ese momento presentan mayor adaptación. El proceso forma una búsqueda de genera y prueba por lote. La popularidad de los algoritmos genéticos se debe a factores que incluyen:

- La evolución puede verse como un método robusto de adaptación, de gran éxito en los sistemas biológicos.

- Los algoritmos genéticos pueden buscar en espacios de hipótesis que contienen elementos en compleja interacción, de tal forma que la influencia de cada elemento sobre la medida de adaptación global de las hipótesis, sea difícil de modelar.
- Los algoritmos genéticos son fácilmente paralelizables y pueden por tanto, sacar ventaja de la reducción del costo de hardware para cómputo en paralelo.

1. Algoritmos genéticos

El problema que confrontan los algoritmos genéticos consiste en identificar dentro de un espacio de hipótesis candidato, la mejor, donde la mejor hipótesis es aquella que optimiza una medida numérica predefinida para el problema, llamada adaptación (*fitness*) de la hipótesis. Por ejemplo, si la tarea de aprendizaje es el problema de aproximar una función desconocida dado un conjunto de entrenamiento de entradas y salidas, la adaptación puede definirse como la precisión de la hipótesis sobre el conjunto de entrenamiento (porcentaje de éxitos al predecir la salida). Si la tarea de aprendizaje tiene la forma de un juego, la adaptación puede medirse en términos del porcentaje de partidas ganadas.

Aunque los detalles de implementación varían entre diferentes algoritmos genéticos, todo comparten en general la siguiente estructura: El algoritmo opera iterativamente, actualizando un conjunto de hipótesis llamada *población*. En cada iteración, todos los miembros de la población son evaluados de acuerdo a una *función de adaptación*. Una nueva población es generada, seleccionando probabilísticamente los individuos de mayor adaptación en la población presente. Algunos de estos individuos pasan intactos a la siguiente generación. Otros son seleccionados para crear una nueva prole, aplicando operaciones genéticas como el *cruce* y la *mutación*.

El prototipo de un algoritmo genético se muestra en el cuadro 1. Las entradas de este algoritmo incluyen una función de adaptación para evaluar los candidatos a hipótesis, un umbral definiendo el nivel aceptado de adaptación para dar por terminado el algoritmo, el tamao que debe mantener la población, y los parámetros necesarios para determinar cómo evoluciona la población, esto es, la fracción de la población que será remplazada en cada generación y la tasa de mutación presente.

Observen que cada iteración de este algoritmo produce una nueva generación de hipótesis, con base en la población actual de hipótesis. Primero, un cierto número de hipótesis en la población $((1-r)\times p)$ es *seleccionado* probabilísticamente para su inclusión en la prole. La probabilidad de una hipótesis $h_i \in pob$ de ser seleccionada está dada por:

$$Pr(h_i) = \frac{adaptacion(h_i)}{\sum_{j=1}^{p} adaptacion(h_j)}$$

Por lo tanto, la probabilidad de que una hipótesis sea seleccionada es proporcional a su propio índice de adaptación, e inversamente proporcional a la

```
fun AG(adaptación, umbral, p, r, m)
    input:  adaptación : función de adaptación;
            umbral : adaptación suficiente para parar;
            p : número de hipótesis en la población;
            s : fracción de la población a substituir;
            m : tasa de mutación;
    static: pob: población;
            pobAux: caché para recalcular población;
    output: maxh: la hipótesis con mejor adaptación;

    pob ← generadorHipótesis(p);
    foreach h in pob do
            adaptación(h);
    endforeach
    while maxAdaptación(pob) < umbal do
          pobAux ← selección((1 − r) × p, pob);
          pobAux ← pobAux ∪ cruce((r × p)/2, pob);
          pobAux ← mutar(m, pobAux);
          pob ←pobAux;
          foreach h in pob do
                  adaptación(h);
          endforeach
    endwhile
    return maxh ← maxAdaptación(pob);
endfun
```

Cuadro 1: *Prototipo de un algoritmo genético.*

adaptación de las hipótesis concurrentes en la misma población.

Una vez que estos miembros de la población son seleccionados para ser incluidos en la prole, el resto de los miembros de ésta se genera usando el operador de *cruce*. Este operador, que será definido en detalle en la próxima sección, selecciona $(r \times p)/2$ pares de hipótesis en la población y genera dos descendientes para cada par, combinando porciones de ambos padres[1]. Los padres son elegidos probabilísticamente con la $Pr(h_i)$ definida como hasta ahora. En este momento, la prole tiene el tamao de población deseado p, así que solo resta seleccionar m elementos de la población para aplicar la operación de *mutación*. Estos elementos son elegidos al azar y los cambios efectuados en ellos son también aleatorios. Algunos enfoques de algoritmos genéticos practican un elitismo al evitar explícitamente que las mejores hipótesis encontradas hasta ese momento, puedan

[1]Aunque en los sistemas biológicos que inspiran estos algoritmos, toda prole, al igual que usted y yo, tiene padre y madre, aquí sólo se habla de padres. Para evitar problemas de género, consideren que el mencionar únicamente a los padres, no significa que nuestras hipótesis no tengan madre. Y consideren que aclaraciones como ésta, nunca son ociosas.

mutar.

Este algoritmo genético lleva a cabo una búsqueda por barrido (*beam*), paralela y aleatoria de hipótesis que tienen un buen desempeo de acuerdo a la función de adaptación.

1.1. Representación de hipótesis

Las hipótesis en un algoritmo genético son representadas por lo general como cadenas de bits, de forma que puedan ser fácilmente manipuladas por los operadores de cruce y mutación. Las hipótesis representadas de esta manera pueden ser bastante complejas. Por ejemplo, conjuntos de reglas Si-Entonces son fáciles de representar, eligiendo un código para las reglas que asigna subcadenas específicas de la hipótesis a las pre-condiciones y post-condiciones de la regla [6].

Primero consideren como podemos usar una cadena de bits para describir una restricción sobre un atributo, por ejemplo, el atributo *cielo* con sus tres valores posibles: *soleado*, *nublado*, *lluvia*. Una manera obvia de codificar una restricción sobre este atributo, consiste en usar una cadena de 3 bits, en donde cada posición de la cadena corresponde a un valor en particular. Colocar un 1 en alguna posición, indica que el atributo puede tomar el valor correspondiente. Por ejemplo, la cadena `010`, representa la restricción *cielo=nublado*. De manera similar, la cadena `011` representa la restricción más general *cielo=nublado* $\vee$ *cielo=lluvia*. Observen que `111` representa la restricción más general posible, indicando que no nos importa que valor tome el atributo *cielo*.

Dado este método para representar restricciones sobre atributos, las conjunciones de restricciones sobre múltiples atributos pueden representarse por concatenación. Por ejemplo, consideren un segundo atributo *viento*, con valores posibles *fuerte* y *débil*. La precondición de una regla como:

$$(cielo = nublado \vee lluvia) \wedge (viento = fuerte)$$

puede representarse por la siguiente cadena de bits de longitud 5:

cielo	*viento*
`011`	`10`

Las post-condiciones, como *jugar-tenis? = si*, pueden representarse de la misma manera. La regla completa puede describirse concatenando también la cadena que representa la post-condición. Por ejemplo, la regla: Si *viento = fuerte* Entonces *jugar-tenis? = si*, puede representarse como:

cielo	*viento*	*jugar-tenis?*
`111`	`10`	`10`

Los primeros tres bits indican que no nos importa el valor del atributo *cielo*; los siguientes dos describen la restricción sobre *viento*; los dos últimos bits representan la post-condición de la regla. Aquí asumimos que *jugar-tenis?* puede

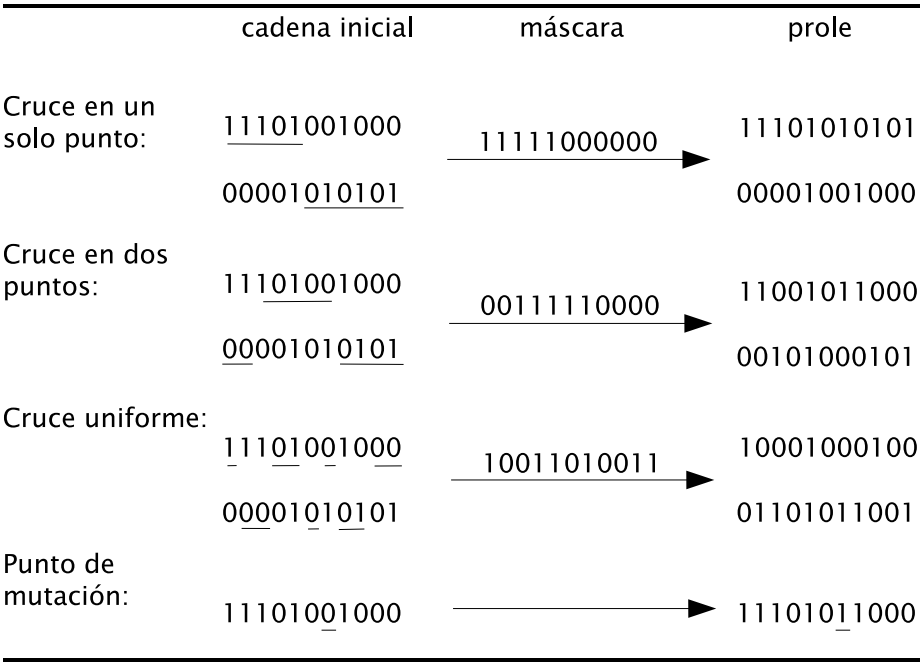

Figura 1: *Operadores genéticos comunes. La máscara determina la contribución de cada padre. La mutación sólo crea un descendiente.*

tomar valores *si* o *no*. Esto nos da una representación donde las reglas que nos interesan se codifican como cadenas de bits de longitud fija.

Al disear el código binario para un espacio de hipótesis, es útil considerar que toda cadena de bits que es sintácticamente valida, representa una hipótesis bien definida. Por ejemplo, en el código anterior, la cadena `111 10 11`, representa una regla cuya post-condición no establece restricciones sobre el atributo *jugar-tenis?*. Para evitar esta posibilidad, se puede usar un código diferente, por ej., para el atributo *jugar-tenis?* usar un solo bit que indique los valores *si* o *no*; o bien, se pueden alterar los operadores genéticos para evitar explícitamente la construcción de tales cadenas de bits; o simplemente asignar un valor de adaptación muy bajo a estas cadenas.

1.2. Operadores genéticos

La generación de la prole en un algoritmo genético está determinada por un conjunto de operadores que combinan y mutan a los miembros seleccionados de la población actual. Los operadores típicos se muestran en la figura 1 y corresponden a versiones idealizadas de las operaciones genéticas que operan en la evolución biológica.

El operador de *cruce*, produce dos nuevos descendientes, a partir de dos

cadenas de bits, copiando bits seleccionados de cada padre, en los hijos. El bit en la posición i de cada hijo, es copiado del bit en la posición i de uno de los padres. La decisión de que padre contribuye con el bit i está determinada por una cadena adicional llamada *máscara de cruce*. Como ejemplo, consideren el operador de cruce en un sólo punto de la figura 1, la máscara de cruce para este operador es `111110000000` lo cual significa que los descendientes tomaran los primeros 5 bits de uno de los padres y el resto del otro padre, como lo muestra la figura. En el *operador de cruce en un sólo punto*, la máscara siempre se construye comenzando por n posiciones adyacentes de 1s, seguida de la cantidad necesaria de 0s para completar la cadena. Cada que se aplica el operador, el punto de cruce n es elegido aleatoriamente, y la máscara correspondiente es creada y aplicada.

En el *cruce de dos puntos*, la prole es creada substituyendo un segmento intermedio de uno de los padres, por el segmento intermedio del otro padre. Esto es, la máscara comienza con una secuencia de n_0 0s, seguida de una secuencia de n_1 1s, y al final, los 0s necesarios para completar la cadena. Cada vez que un operador de cruce en dos puntos es aplicado, la máscara es generada a partir de la elección de dos enteros n_0 y n_1. En el ejemplo de la figura 1, $n_0 = 2$ y $n_1 = 5$. De nuevo, los dos hijos se logran intercambiando para cada uno, el papel de los padres.

En el *cruce uniforme*, una máscara generada aleatoriamente, se aplica por igual a ambos padres para generar los dos hijos. Cada bit de la máscara se genera independientemente de los otros. El operador de *mutación* produce una pequea variación aleatoria sobre una cadena, cambiando el valor de un bit en particular.

Algunos algoritmos genéticos emplean operadores especializados para una representación de hipótesis en particular. Por ejemplo, John Greffensette et al. [4] describe un sistema para aprender conjuntos de reglas para controlar un robot. Además del cruce y la mutación, este sistema usa un operador para especializar las reglas.

1.3. Función de adaptación y Selección

La función de adaptación (*fitness*) define el criterio para ordenar las hipótesis potenciales y para seleccionarlas probabilísticamente, para su inclusión en la siguiente generación de la población. Si la tarea consiste en aprender reglas de clasificación, entonces la función de adaptación contiene un componente para evaluar la precisión de las hipótesis sobre un conjunto de entrenamiento dado. Otros criterios pueden ser tomados en cuenta como la complejidad o la generalidad de una regla. Si las cadenas de bits son interpretadas como procedimientos, i.e., reglas de producción, la función de adaptación debe medir el desempeo global de esas reglas y no únicamente su precisión individual.

En el algoritmo del cuadro 1, la probabilidad de que una hipótesis sea seleccionada está dada por el radio de su adaptación y la adaptación de los demás miembros de la población. Este método se conoce como *selección proporcional a la adaptación*, o selección de *ruleta*. Otros métodos de selección han sido propuestos, por ejemplo, la selección por *torneo*, donde dos hipótesis son elegi-

das aleatoriamente de la población, entonces, con una probabilidad p definida previamente, la hipótesis más adaptada es seleccionada, y con probabilidad $(1 - p)$ la hipótesis menos adaptada es seleccionada. De esta forma se obtiene una población más diversificada [3]. En otro método llamado selección por *rango*, las hipótesis son ordenadas de acuerdo a su adaptación y la probabilidad de una hipótesis de ser seleccionada es proporcional a su rango entre las hipótesis ordenadas.

2. Un ejemplo ilustrativo

Un algoritmo genético puede verse como un método de optimización general, que busca en un gran espacio de candidatos, a los elementos que tienen mejor desempeo con respecto a la función de adaptación. Aunque no garantizan encontrar el elemento óptimo, los algoritmos genéticos generalmente tienen éxito en encontrar un elemento con alta adaptación. Para ilustrar el uso de los algoritmos genéticos en el aprendizaje de conceptos, presentare brevemente al sistema GABIL [2] que usa algoritmos genéticos para aprender conceptos booleanos representados como un conjunto disyuntivo de reglas proposicionales. En experimentos, GABIL ha resultado tener tanta precisión como C4.5 y el sistema de aprendizaje de reglas AQ14.

El algoritmo usado por GABIL es exactamente el mostrado en el cuadro 1. En los experimentos reportados por Kenneth DeJong [2], el parámetro r que determina el porcentaje de la población que será remplazada por cruce, fue fijado en 0.6. El parámetro m que determina la tasa de mutación, fue fijado en 0.001. Estos son valores típicos para ambos parámetros. El tamao de la población varió entre 100 y 1000, dependiendo de la tarea de aprendizaje específica en cada experimento.

Las hipótesis en GABIL se representan como un conjunto disyuntivo de reglas proposicionales, codificadas como se describió en la sección anterior (pág. 4), sólo que el atributo objetivo es codificado por un sólo bit. Para representar el conjunto de reglas, las cadenas representando reglas individuales son concatenadas. Por ejemplo, consideren un espacio de hipótesis donde las pre-condiciones de las reglas son conjunciones restricciones sobre dos atributos a_1 y a_2. El atributo objetivo es c. Entonces, la hipótesis que consta de estas dos reglas: Si $a_1 = T \wedge a_2 = F$ Entonces $c = T$; Si $a_2 = T$ Entonces $c = F$, se representa por la cadena:

a_1	a_2	c	a_1	a_2	c
10	01	1	11	10	0

Observen que la longitud de la cadena de bits crece con el número de reglas en la hipótesis. Esta longitud variable requiere una ligera modificación en la definición de operador de cruce. El operador de cruce usado por GABIL es una extensión estándar del operador de cruce en dos puntos, descrito en la figura 1. En particular, dada la longitud variable de la cadena de bits representando las

hipótesis, el operador de corte debe estar restringido a aplicarse bajo las siguientes condiciones: Dos padres son seleccionados, luego dos puntos de cruce son elegidos aleatoriamente sobre el primer padre. Sea $d_1(d_2)$ la distancia del punto más a la izquierda (más a la derecha) al límite de la regla inmediatamente a la izquierda. Los puntos de cruce en el segundo padre se fijan ahora aleatoriamente con la restricción de que deben tener los mismos valores $d_1(d_2)$. Por ejemplo, si los dos padres son:

	a_1	a_2	c	a_1	a_2	c
$h_1 =$	10	01	1	11	10	0

y:

	a_1	a_2	c	a_1	a_2	c
$h_2 =$	01	11	0	10	01	0

y los puntos de cruce para el primer padre son los bits 1 y 8:

	a_1	a_2	c	a_1	a_2	c
$h_1 =$	1[0	01	1	11	1]0	0

entonces $d_1 = 1$ y $d_2 = 3$, por lo que los pares de puntos de cruce permitidos para la hipótesis h_2 son: $\langle 1, 3\rangle$, $\langle 1, 8\rangle$ y $\langle 6, 8\rangle$. Si el par $\langle 1, 3\rangle$ es elegido:

	a_1	a_2	c	a_1	a_2	c
$h_2 =$	0[1	1]1	0	10	01	0

Por lo que la prole resultante de estas dos hipótesis es:

	a_1	a_2	c
$h_3 =$	11	10	0

y:

	a_1	a_2	c	a_1	a_2	c	a_1	a_2	c
$h_4 =$	00	01	1	11	11	0	10	01	0

Como se puede ver en el ejemplo, este operador de cruce permite que la prole contenga diferente número de reglas que sus padres, y al mismo tiempo garantiza que todas las cadenas de bits generadas de esta forma, representen conjuntos de reglas bien definidos.

La función de adaptación para cada hipótesis se basa en su precisión como clasificador sobre un conjunto de ejemplos de entrenamiento. En particular, la función usada para medir la adaptación es:

$$adaptacion(h) = (correcto(h))^2$$

donde $correcto(h)$ es el porcentaje de ejemplos bien clasificados por la hipótesis h.

2.1. Extensiones

Kenneth DeJong explora dos extensiones interesantes al sistema GABIL. En una serie de experimentos, se estudia la inclusión de dos nuevos operadores genéticos, motivados por la operación de generalización tan común en el aprendizaje de conceptos. El primero de estos operadores *agregaAlternativa*, generaliza una restricción sobre un atributo, cambiando un 0 por un 1 en la sub cadena que corresponde al atributo. El segundo operador *eliminaCondición*, lleva a cabo una generalización más drástica, remplazando todos los bits de una sub cadena por 1s, esto es, elimina por completo la restricción en un atributo dado. En una serie de tareas de aprendizaje sintéticas [2], aplicando el segundo operador con probabilidad 0.6, el sistema llega a una precisión de 95.2 % contra un 92.1 % de GABIL clásico.

En estos experimentos, los dos operadores fueron aplicados con la misma probabilidad a cada una de las hipótesis en cada generación. En otra serie de experimentos, las cadenas de bits representando las hipótesis fue extendida para incluir 2 bits que determinan cual de esos operadores será aplicado a la hipótesis. Estos dos bits definen entonces parte de la estrategia de búsqueda del algoritmo genético y puesto que forman parte de la hipótesis, son sujetos de modificación por los operadores genéticos. Aunque este experimento dió resultados ambivalentes, mejoras en algunos problemas y baja de desempeo en otros, es sumamente interesante porque sugiere la posibilidad de que el algoritmo genético evolucione su propio método de búsqueda de hipótesis.

3. Espacio de hipótesis

Los algoritmos genéticos emplean una búsqueda al azar por barrido para encontrar la mejor hipótesis. Observen que esta forma de buscar es diferente de la búsqueda que llevan a cabo los otros algoritmos que hemos visto en el curso. Debido que el algoritmo genético puede moverse abruptamente en el espacio de hipótesis posibles, por ejemplo, cuando la prole difiere totalmente de sus padres, es menos probable que el algoritmo caiga en máximos locales.

Uno dificultad práctica al usar algoritmos genéticos es conocida como *crowding*. En este fenómeno, un individuo que es más apto que otros miembros de la población, comienza a reproducirse rápidamente de tal forma que copias de este individuo, o bien de individuos muy parecidos, ocupan una fracción importante de la población. El efecto negativo de este fenómeno es que reduce la diversidad de la población, haciendo la búsqueda más lenta. Algunas técnicas para evitar el *crowding* incluyen: Usar un criterio de selección por torneo o por rango, en lugar de la selección basada en adaptación; o bien reducir el índice de adaptación de un individuo, si se detecta la presencia de individuos similares en la población (*fitness sharing*); y restringir que individuos pueden combinarse para producir prole. De esta manera, si sólo se permite a individuos semejantes reproducirse, se alienta la creación de especies o grupos de individuos similares. Otro enfoque para restringir la reproducción es considerar que los elementos de la población se

encuentran distribuidos en un espacio, de tal forma que sólo individuos cercanos entre si, pueden reproducirse.

3.1. Evolución de la población y el teorema de esquema

¿Puede uno caracterizar matemáticamente la evolución de la población en un algoritmo genético? El *teorema del esquema* de John Holland [5] provee esa caracterización. El teorema esta basado en el concepto de esquemas, o patrones, que describen conjuntos de cadenas de bits. Un esquema es cualquier cadena compuesta de 1s, 0s y *s. Por ejemplo, el esquema `0*10` representa el conjunto de cadenas {`0010, 0110`}. Una cadena de bits individual, puede verse como representante de cada uno de los esquemas que satisface. Por ejemplo, la cadena `0010` puede verse como representante de 2^4 esquemas diferentes, incluyendo `00**`, `0*10`, `****`, etc. De manera similar, una población de cadenas de bits, puede verse en términos del conjunto de esquemas que satisface y el número de individuos asociados con cada uno de esos esquemas.

El teorema del esquema caracteriza la evolución de una población en un algoritmo genético en términos del número de elementos representando cada esquema. Sea $m(s,t)$ el número de elementos del esquema s al tiempo t. El teorema del esquema describe el valor esperado de $m(s,t+1)$ en términos de $m(s,t)$ y otras propiedades del esquema, la población, y los parámetros del algoritmo genético.

Estamos pues interesados en calcular el valor esperado de $m(s,t+1)$, identificado como $E[m(s,t+1)]$, utilizando la distribución de probabilidad dada por la función de selección, que puede formularse como sigue:

$$\begin{aligned} Pr(h) &= \frac{f(h)}{\sum_{i=1}^{n} f(h_i)} \\ &= \frac{f(h)}{n\bar{f}(t)} \end{aligned}$$

Ahora, si seleccionamos un miembro de la nueva población de acuerdo con esta distribución de probabilidad, entonces la probabilidad de que un representante del esquema s sea seleccionado es:

$$\begin{aligned} Pr(h \in s) &= \sum_{h \in s \cap p_t} \frac{f(h)}{n\bar{f}(t)} \\ &= \frac{\hat{u}(s,t)}{n\bar{f}(t)} m(s,t) \end{aligned}$$

Esto sigue del hecho de que por definición:

$$\hat{u}(s,t) = \frac{\sum_{h \in s \cap p_t} f(h)}{m(s,t)}$$

Esto nos da la probabilidad de que una sola hipótesis seleccionada por el algoritmo genético, sea un ejemplar del esquema s. Por lo tanto, el número esperado de ejemplares de s que resulten de n selecciones independientes que crean una generación enteramente nueva, es sólo n por esta probabilidad.

La siguiente expresión

$$E[m(s,t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)$$

establece que el número esperado de ejemplares del esquema s en la generación $t+1$, es proporcional a la adaptación promedio $\hat{u}(s,t)$ de los ejemplares de este esquema al tiempo t, e inversamente proporcional a la adaptación promedio $\bar{f}(t)$ de todos los miembros de la población al tiempo t. Por lo tanto, podemos esperar que aquellos esquemas con adaptación sobre el promedio estar representados con más frecuencia en las generaciones sucesivas. Si vemos al algoritmo genético como un proceso de búsqueda paralela virtual en un espacio de esquemas posibles, la expresión anterior nos dice que los esquemas con mayor adaptación tendrán mayor influencia con el tiempo.

Mientras que el análisis anterior considera solo el paso de selección en el algoritmo genético, los pasos de cruce y mutación deben ser considerados también. El teorema del esquema considera sólo la posible influencia negativa de estos operadores genéticos y considerando sólo el caso de cruce en un solo punto, por ejemplo, la mutación aleatoria puede disminuir el número de representantes de s, independientemente de $\hat{u}(s,t)$. El teorema del esquema completo, provee entonces una cota inferior sobre la frecuencia esperada del esquema s:

$$E[m(s,t+1)] \geq \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t) \left(1 - p_c \frac{d(s)}{l-1}\right) (1-p_m)^{o(s)}$$

Aquí, p_c es la probabilidad de que el operador de corte en un solo punto, sea aplicado a un individuo arbitrario; y p_m es la probabilidad de que un bit arbitrario de un individuo arbitrario, sea mutado. $o(s)$ es el número de bits definidos en el esquema, esto es bits diferentes de $*$. $d(s)$ es la distancia entre los bits definidos más a la izquierda y más a la derecha en s. El primer término de esta expresión describe, como vimos, el efecto de la operación de selección. El término medio describe el efecto del operador de corte en un solo punto, en particular describe la probabilidad de que un representante del esquema s lo siga siendo, después de aplicar el operador de corte. El último término expresa la probabilidad de que un representante de s, lo siga siendo luego de una operación de mutación. Observen que los efectos de los operadores de cruce y mutación se incrementan con el número de bits definidos $o(s)$ en el esquema y con la distancia $d(s)$ entre los bits definidos. Entonces, el teorema del esquema puede interpretarse como: los esquemas más ajustados tenderán a ser más influyentes, especialmente los esquemas que contienen un pequeo número de bits definidos y especialmente cuando esos bits son cercanos dentro de la cadena de bits.

El teorema del esquema es quizá la caracterización más citada de la evolución de una población en un algoritmo genético. El teorema es incompleto en

el sentido que falla al predecir los supuestos efectos positivos del cruce y la mutación. Análisis más recientes se basan en modelos de Markovy modelos de mecánica estadística.

4. Modelos de evolución y aprendizaje

En muchos sistemas naturales, los organismos individuales aprenden a adaptarse significativamente durante su vida. Al mismo tiempo, diversos procesos biológicos y sociales permiten que las especies se adapten en la escala de tiempo de varias generaciones. Una pregunta interesante en esto es ¿Qué relación existe entre el aprendizaje individual y el "aprendizaje" provisto por la evolución?

4.1. La evolución según Lamarck

Lamarck propuso en el siglo XIX, que la evolución sobre muchas generaciones, estaba directamente influenciada por las experiencias de los organismos individuales durante su vida. En particular propuso que las experiencias de un organismo afectaban directamente el marcaje genético de su prole. Si un individuo aprendía durante su vida a evitar cierto alimento tóxico, podría pasar esta característica genéticamente a sus descendientes, que por lo tanto, no tendrían necesidad de aprender esta propiedad. Esta conjetura es atractiva porque podría presumiblemente permitir un progreso evolutivo más eficiente que el proceso genera-prueba de los algoritmos genéticos, que ignoran la experiencia ganada durante el tiempo de vida de los individuos. A pesar de lo atractivo de la teoría, la evidencia científica actual contradice el modelo de Lamarck. El punto de vista aceptado actualmente es que el marcaje genético de un individuo es de hecho, inalterable por la experiencia de vida de sus padres biológicos. Sin embargo, y a pesar de esta contradicción, algunos estudios computacionales han mostrado que los procesos Lamarckianos pueden en algunos casos, mejorar la efectividad de los algoritmos genéticos.

4.2. El efecto Baldwin

James M. Baldwin [1] describe un mecanismo que sugiere que el aprendizaje individual altera el curso de la evolución. El efecto Baldwin está basado en las siguientes observaciones:

- Si una especie está evolucionando en un ambiente cambiante, habrá una presión evolutiva en favor de los individuos con la capacidad de aprender durante su tiempo de vida.
- Aquellos individuos que son capaces de aprender muchos rasgos, dependen menos de su código genéticos de rasgos alambrados. Como resultado, estos individuos pueden soportar un pool genético más diverso, basándose en el aprendizaje individual para resolver los rasgos perdidos o no optimizados

de su código genético. Este pool genético diverso, puede a su vez, soportar una adaptación por evolución más rápida.

El número especial de *Evolutionary Computation* [7], contiene varios artículos sobre el efecto Baldwin en los algoritmos genéticos.

5. Cómputo paralelo

Los algoritmos genéticos pueden implementarse en paralelo de manera natural. Diversos enfoques de inmplementación en paralelo han sido estudiados. Los enfoques llamados de grano grueso (*coarse grain*) suelen dividir la población entre grupos distintos de individuos llamados *demes*. Cada deme es asignado a un nodo de computación y un algoritmo genético estándar se aplica en cada nodo. Comunicación y fertilización entre los demes ocurre con menor frecuencia que al interior de ellos. La transferencia entre entre los demes se da por un proceso de migración. Una de las ventajas de estos enfoques es que reducen en gran medida el fenómeno de *crowding* que es común en versiones no paralelas de los algoritmos genéticos. La paralelización de grano fino, generalmente asigna un procesador a cada miembro de la población. Las combinaciones se dan sólo entre vecinos y diferentes tipos de vecindad son propuestas.

Referencias

[1] J.M. Baldwin. A new factor in evolution. *American Naturalist*, 3:441–451, 536–553, 1896. www.santafe.edu/publications/Bookinfo/baldwin.html.

[2] K.A. De Jong, W.M. Spears, and D.F. Gordon. Using genetic algorithms for concept learning. *Machine Learning*, 13:161–188, 1993.

[3] D. Goeldberg and K. Deb. mGA in C: A messy genetic algorithm in C, 1991.

[4] J.J. Grefenstette. Lamarckian learning in multi-agent environments. In R. Belew and L. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 303–310, San Mateo, CA., EEUU, 1991. Morgan Kaufman.

[5] J.H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, 1975. Reimpreso por MIT Press en 1992.

[6] J.H. Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2. Morgan Kaufmann, San Mateo, CA., EEUU, 1986.

[7] P.D. Turney, D. Whitley, and R. Anderson. The baldwin effect. *Evolutionary Computation*, 4(3), 1997. www.mitpress.mit.edu/jrnls-catalogue/evolution-astracts/evol.html.