Recorrido a través de la Programación Genética

Guillermo Choque Aspiazu
gchoque@umsanet.edu.bo

Resumen
En este articulo se realiza un recorrido a través de los conceptos iniciales de la programación genética, se utiliza la noción de los árboles de análisis para presentar la manera de discretizar los problemas en su paso hacia su solución mediante paradigmas computacionales, se emplean los estudios relativos al lenguaje LISP en analogía con los árboles de análisis, útiles necesarios para la comprensión de la manera en la que se enfoca el desarrollo de soluciones en la programación genética.
Palabras clave: *programación, algoritmos genéticos, árboles de análisis, adaptabilidad, apareamiento, mutación.*

1. INTRODUCCIÓN

A lo largo del período de evolución de los seres vivos, se fueron seleccionando conductas que son adecuadas para la supervivencia. Las especies que sobreviven son aquellas que están mejor adaptadas al ambiente que les rodea. En la lucha por sobrevivir, los individuos mejor adaptados sobreviven más fácilmente y por tanto pueden reproducirse y transferir a sus descendientes las cualidades beneficiosas que les permitieron estar mejor adaptados, lo cual constituye el proceso de selección natural explicado por Charles Darwin en su libro "*El origen de las especies*". Todos aquellos rasgos que faciliten la supervivencia del individuo tenderán a mantenerse, mientras que lo que constituya una dificultad o debilidad para la adaptación tenderá a desaparecer, pues su poseedor no tendrá oportunidad de legarlo a su descendencia.

Según Mitchell (1997), la Programación Genética (PG)[1] es "*una forma de computación evolutiva en la que los individuos de la población son programas de computadora en lugar de cadenas de bits*". Al igual que en los algoritmos genéticos, la población se selecciona siguiendo el principio darwiniano de supervivencia del más apto.

La PG es una materia de estudio bastante reciente, pese a que es posible encontrar la idea de su origen ya en los años 50. Efectivamente, en esa década Arthur Samuel[2] planteó la cuestión que ha llevado al desarrollo de la computación evolutiva: ¿cómo pueden los ordenadores aprender a resolver problemas sin ser explícitamente programados?. La PG surgió como una evolución de los Algoritmos Genéticos (AGs). De forma resumida, los AGs son una forma de aprendizaje basada en una evolución simulada. Las hipótesis se describen como cadenas de bits, cuya interpretación depende de la aplicación, que van evolucionando siendo seleccionadas según determinados criterios.

Ya en la Primera Conferencia sobre AGs, desarrollada en la Universidad de Carnegie Mellon, se puede encontrar dos artículos que, sin usar explícitamente el nombre de PG, pueden ser considerados como precursores en la materia: "*A Representation for the Adaptive Generation of Simple Sequential Programs*" [Cramer, 1985] y "*Using the Genetic Algorithm to Generate Lisp Source Code to solve the Prisoner's Dilemma*" [Fujiki & Dickinson, 1987].
Las metas de la PG constituyen el uso de los conceptos de la genética y de la selección natural de Darwin para generar y evolucionar

[1] Termino introducido por el considerado padre de la programación genética: John R. Koza.

[2] Arthur Samuel (1901-1990) fue un pionero de la investigación en Inteligencia Artificial. Desde 1949 a fines de los años 60, realizó su mejor trabajo desarrollando métodos de aprendizaje de las computadoras a partir de su experiencia. Su vehículo para lograrlo fue el juego del ajedrez.

programas computacionales completos. La PG se asemeja ampliamente a los AGs en términos de su algoritmo básico. Las nociones de mutación, apareamiento y adaptabilidad son esencialmente las mismas, solamente que con la PG se requiere un cuidado especial cuando se utilizan esas operaciones. Mientras que los AGs están interesados en la modificación de strings de tamaño fijo, usualmente asociados con parámetros a una función, la PG está interesada actualmente en la creación y manejo de la estructura de un programa, de tamaño no fijo. Como resultado, la PG es un tópico mucho más complejo y dificultoso.

En la presente propuesta se realiza un recorrido a través de los conceptos iniciales de la PG, se utiliza la noción de los árboles de análisis para presentar la manera de reducir en términos computacionales los problemas en su paso hacia su solución, se utiliza los estudios relativos al lenguaje LISP[3] en analogía con los árboles de análisis, útiles necesarios para la comprensión de la manera en la que se enfoca el desarrollo de soluciones en la programación genética.

2. ÁRBOLES DE ANÁLISIS

La representación del conocimiento es de gran importancia en la PG. En [Koza, 1992], por ejemplo, es posible encontrar un capítulo completo dedicado a ello. Generalmente, los programas manipulados en PG están representados por árboles que corresponderían a un árbol de análisis[4] del programa. Cada llamada a función se representaría por un nodo en el árbol, y los argumentos de la función vendrían dados por sus nodos descendientes

En términos detallados, uno de los requerimientos principales en los cuales se debe concordar cuando se generan posibles programas es que los mismos deben ser validos en su sintaxis. Con algunos lenguajes tales como el C++, es casi imposible realizar esto en su forma genérica. Como resultado, el programa debe ser afinado en una representación más elegante. Los árboles de análisis son representaciones simples y homogéneas de los programas computacionales. En el proceso de compilación de un programa de alto nivel, como uno escrito en C o C++, el código es volcado en un árbol de análisis para facilitar su manejo por parte de la computadora. Como resultado de esta aplicación elegante, los árboles de análisis constituyen la lengua nativa de la PG permitiendo que los operadores genéticos puedan ser aplicados de manera sencilla y coherente. Otro de los beneficios es que todos los programas LISP pueden ser fácilmente escritos como un árbol de análisis y viceversa [Garner, 2000].

Si se considera la evaluación de una expresión simple: **x + 8**. En términos de un árbol de análisis, puede ser escrita como:

```
  +
 / \
x  8
```

Una expresión más compleja tal como **(x * y + (56 + 3)) - (8 / (dist x y))** puede ser escrita de la siguiente manera:

```
          -
      /       \
    +           /
  / \         / \
 *    +     8   dist
/ \  / \         / \
x y 56 3        x y
```

Cuando se observa la expresión en forma de árbol, solamente se debe tener cuidado acerca de una cosa referida a la sintaxis: que ocurre cuando algún elemento es o no un nodo hoja.

---

[3] Lenguaje para el procesamiento de listas de palabras, se reconoce como el lenguaje de programación de la inteligencia artificial, en este lenguaje se desarrollaron los primeros programas genéticos.

[4] La terminología genérica refiere a los mismos como: Parse Trees.

Cada función que requiere un parámetro (tal como la suma, multiplicación o encontrar la distancia entre dos puntos) constituye la cabeza de un subárbol. Cada nodo hoja será una variable, una constante o una función sin parámetros. Los nodos hoja no dependen de ningún valor alternativo para sus valores y son referenciados como nodos terminales. Ahora se tiene un camino consistente y simple para expresar un programa. Como se ha mencionado de manera previa, Los programas LISP son básicamente análogos a los árboles de análisis. El primer ejemplo **x + 8** puede ser escrito en la notación prefija del LISP de la siguiente manera: **(+ x 8)**. El segundo ejemplo puede ser escrito como **(-(+(* x y)(+ 56 3))(/8(dist x y)))**. Como puede observarse, las listas anidadas en los subárboles y en las expresiones LISP pueden ser intercambiadas de manera sencilla, especialmente si se cuenta previamente con una pequeña practica.

2.1. Propiedades

De acuerdo a Garner (2000), existen dos propiedades que son necesarias y que deben ser consideradas para que todos los elementos puedan interactuar adecuadamente unos con otros. Primero, se debe contar con una clausura. La clausura requiere que alguna función que toma un valor sea capaz de manejar cualquier valor que la misma pueda proporcionar. Si el análisis estuviera limitado a las funciones aritméticas estándar: **{+, -, *, /}** y a los valores: **{0, 1, 2}** como terminales, no se tiene una clausura. Es posible que se presente el siguiente árbol de análisis: **(/2 0)** (recuerde que es lo mismo que (2/0) en la notación infija). Si esto fuera verdadero, el programa abortaría su ejecución con el error división-por-cero. La clausura puede ser forzada, en muchas situaciones, redefiniendo la función asociada a la misma. Como ejemplo, la función divide puede ser escrita de modo tal que retorne siempre el numero cero cuando se recibe el error división-por-cero. En otro caso, si se contara con una función unaria que necesita tomar dos opciones, se podría simplemente ignorar la segunda función, que es una solución adecuada, pero no elegante, a las excepciones del problema.

La segunda propiedad que es necesario observar es que la combinación de funciones y terminales que se está utilizando sea suficiente y adecuada para resolver el problema. Si se está limitado a las funciones: **{+, -}** y los enteros **{1, 2, 3, 4, ...}**, será imposible encontrar una función que implemente **log(x)**. A lo mejor solamente se pueda obtener una aproximación a la función **log(x)**.

3. ALGORITMO

Para la ejecución de un algoritmo de PG es necesario considerar previamente una serie de aspectos relativos al funcionamiento del propio algoritmo. En primer lugar, se establece que el conjunto de terminales hace referencia a las entradas del programa encargadas de resolver el problema. Las funciones primitivas deben ser apropiadamente definidas: operadores aritméticos, lógicos, de programación, etc. El conjunto de terminales más el conjunto de funciones primitivas deben cumplir el criterio de *suficiencia*, esto es, deben ser capaces de expresar la solución del problema. Por otro lado, cualquier función primitiva debe aceptar como argumento cualquier valor devuelto por cualquier función y cualquier valor que pueda ser usado como terminal. Esto es lo que se denomina *requisito de clausura*. Un algoritmo general que se propone es el siguiente:

Algoritmo PAG()
P1: Crear un estado inicial aleatorio
P2: Evaluar la adaptabilidad
P3: Realizar el apareamiento (recombinación sexual)
P4: Efectuar la mutación
P5: Repetir desde el P2 hasta lograr éxito
Fin Algoritmo PAG.

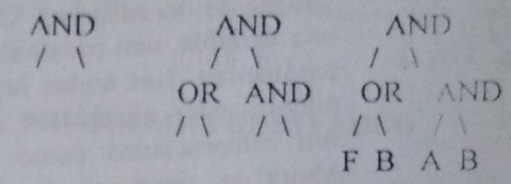
A continuación se presenta un ejemplo del uso de la PG para encontrar una implementación del operador lógico XOR utilizando los operadores lógicos AND, OR y NOT. La Tabla 1 muestra los valores de verdad asociados para dichos operadores lógicos.

Tabla 1. Valores de los operadores lógicos
Fuente: [Mates, 1974]

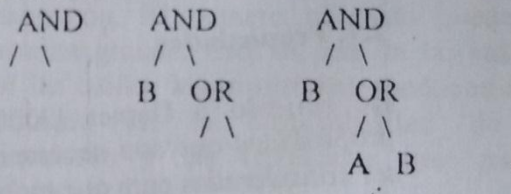
A	B	A AND B	A OR B	NOT A	NOT B	A XOR B
T	T	T	T	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	F	T	T	F

El operador XOR, como puede ser observado, es idéntico al operador regular OR, excepto cuando A y B son verdaderos XOR retorna falso. Los tres operadores lógicos (AND, OR y NOT) y los terminales[5] (A, B, T y F) son suficientes para derivar el operador XOR, las razones detalladas pueden ser encontradas en [Mates, 1974].

Paso 1: Inicializar la Población. Existen dos métodos primarios para seleccionar la población inicial. Para detener la expansión infinita de los árboles iniciales, es necesario establecer un limite máximo de profundidad de los árboles. Por simplicidad, se limita la profundidad de los subárboles a un nivel, sin embargo esto puede variar dependiendo de la aplicación. En el método "completo", los árboles son formados mediante funciones de selección aleatoria que llenan el árbol. De manera pseudo-aleatoria, se elige el operador AND como el nodo padre. Luego, el operador OR como el nodo izquierdo y nuevamente AND como el nodo derecho. Ahora que se ha alcanzado la profundidad derecha, es necesario llenar los nodos hoja con terminales. Los siguientes diagramas muestran la progresión.



En el otro método, las funciones y terminales son seleccionadas al mismo tiempo. La posibilidad de que una terminal pueda mostrarse antes de que la profundidad máxima haya sido alcanzada ocurre para árboles de diferentes formas. Primero el operador AND es seleccionado como el padre, luego el nodo terminal B es seleccionado como el nodo izquierdo, el operador OR para el nodo derecho. Finalmente, las terminales llenan las ranuras restantes después de que se alcanza el máximo.



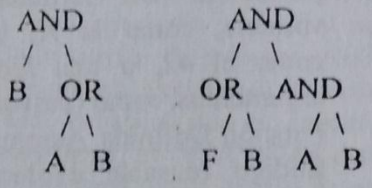
Paso 2: Calcular adaptabilidad. Evaluar la adaptabilidad de un programa requiere el traslado del rendimiento adecuado de un programa en un valor numérico. Usualmente, esto requiere ejecutar el programa un numero de veces diferentes con parámetros diferentes, evaluando la salida en cada etapa. Para los propósitos del ejemplo, se necesita observar el valor de retorno para cuatro casos (Tabla 1). Para programas mas complicados, se debe realizar muchas mas pruebas. Existe un numero variado de maneras para representar la adaptabilidad. La representación más común consiste en tomar la diferencia entre lo que el valor debe representar y el valor que representa. De este modo, el valor de adaptabilidad más pequeño implica el error más pequeño, llegando a constituirse el cero como el mejor valor adaptado. Para el ejemplo del XOR, se retorna simplemente el valor de 1 si una de las pruebas devuelve un resultado erróneo, y un 0 para la respuesta correcta. Luego se

[5] T y F hacen referencia a los valores Verdadero y Falso.

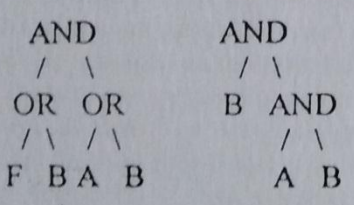
suma los resultados de estas cuatro pruebas para la adaptabilidad.

A continuación, se presenta como se adapta la función AND respecto a la implementación correcta del operador XOR. Esto puede ser observado en la tabla de verdad (Tabla 1), donde cada para de entras de A y B para el operador AND difiere del operador XOR excepto cuando A y B son falsos. De esta manera el operador AND tiene un valor de adaptabilidad de 3.

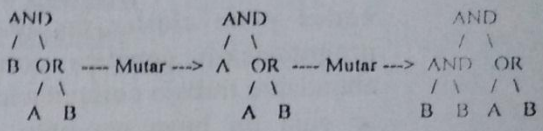
Paso 3: Apareamiento. El apareamiento es simple para los árboles de análisis. Cualquier subárbol o terminal puede ser intercambiado con cualquier otro subárbol o terminal. Como ejemplo, considere los dos programas establecidos previamente:



En un programa actual, se podría recorrer a través de los dos árboles hasta que dos nodos sean seleccionados de manera aleatoria, y los mismos puedan ser intercambiados. Por ejemplo, intercambiando los nodos izquierdos de los dos programas se tiene:



Paso 4: Mutación. La mutación es tan simple como el apareamiento. Cualquier símbolo de la función puede ser reemplazada por cualquier otro símbolo de la función y cualquier terminal puede ser reemplazada por cualquier otra terminal. O cualquier subárbol o terminal puede ser reemplazado con un nuevo subárbol completo generado de la misma manera en la que fue generada la población inicial.



4. PROBLEMAS DERIVADOS

En el recorrido que se realizó a través de la PG, si se considera el numero de posibles funciones, operadores, variables, constantes numéricas y condicionales que se encuentran en un programa promedio; también el numero de permutaciones de esas funciones y variables. El espacio de búsqueda es vasto. Adicionalmente es necesario pensar acerca de cuanto toma ejecutar un programa de complejidad moderada. Ahora se multiplica dicho valor por 500 miembros por población y multiplicando el mismo por 50 o más generaciones y multiplicando por una docena o más de pruebas para determinar la adaptación. El numero obtenido puede ser bastante grande. Esta es una de las razones por las cuales los programas genéticos están limitados en la disponibilidad de los operadores y las terminales que pueden utilizar.

5. CONCLUSIONES

En este articulo se realizó un recorrido a través de los conceptos iniciales de la PG, utilizando la noción de los árboles de análisis para presentar la manera de discretizar los problemas en su paso hacia su solución mediante paradigmas computacionales, se mostraron los estudios relativos a los árboles de análisis en analogía con el lenguaje LISP, útiles necesarios para la comprensión de la manera en la que se enfoca el desarrollo de soluciones. Normalmente el numero obtenido puede ser bastante grande, por lo cual los programas genéticos están limitados en la disponibilidad de los operadores y las terminales que pueden utilizar. Esta es también la razón por la cual John R. Koza, la persona acreditada con el descubrimiento de la programación genética, posee una computadora Pentium con un cluster de 1000

nodos y un cluster de 70 nodos alfa. La programación genética como tal requiere abundante trabajo computacional, aun cuando se elija un buen conjunto de operaciones, terminales y algoritmos de control.

BIBLIOGRAFIA

1. Cramer, N. L.; A representation for the Adaptive Generation of Simple Sequential Programs. *1ª Conferencia sobre Algoritmos Genéticos y sus aplicaciones.* Lawrence Erlbaum Associates, 1985.
2. Fujiki, C. & J. Dickinson; Using the Genetic Algorithm to Generate Lisp Source Code to solve the Prisoner's Dilemma. *2ª Conferencia sobre Algoritmos Genéticos y sus aplicaciones.* Lawrence Erlbaum Associates 1987.
3. Garner, Z.; An Introduction to Genetic Programming. http://www.recurrent.net/ [Acceso: 20 septiembre 2000]
4. Koza, J.R.; *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press 1992.
5. Mates, B; *Lógica Matemática Elemental.* Tecnos, Madrid, 1974.
6. Mitchell, T.; *Machine Learning.* McGraw-Hill 1997.

GLOSARIO

Algoritmo Genético (AG) - Modelo de aprendizaje computacional que usa una metáfora genético-evolutiva. Las implementaciones usan típicamente cadenas de bits de longitud fija para representar la información genética.

Programación Genética (PG) - Algoritmos Genéticos aplicados a programas. La PG es más expresiva que las cadenas de bits de longitud fija de los AGs, aunque los AGs pueden ser más eficientes para algunas clases de problemas.

Apareamiento - Proceso genético mediante el cual se intercambia material genético entre individuos de la población.

Reproducción - Operación genética que origina la creación de una copia exacta de la representación genética de un individuo de la población.

Adaptabilidad - Medida de la aptitud de un individuo para su supervivencia.

Generación - Iteración de la medida de aptitud y la creación de una nueva población por medio de operaciones genéticas.

Conjunto de funciones - Conjunto de operadores usados en la PG. Estas funciones etiquetan los nodos internos (los que no son hojas) de los árboles de análisis que representan los programas de la población. Un ejemplo de conjunto de funciones podría ser {+, -, *}.

Conjunto de terminales - Conjunto de nodos terminales (hojas) de los árboles de análisis que representan los programas de la población. Un terminal puede ser una variable, como la X, un valor constante, como el 42, o una función que no tome argumentos, como (mover-arriba).

Función Definida Automáticamente (FDA) - Código reusable evolucionado de manera automática mediante PG.