4. PROGRAMACION GENETICA CONCEPTOS BASICOS

José J. Martínez P.
josejesusmp@gmail.com
Diciembre 22 1

4.1 Programación Genética

Habíamos hablado de los Algoritmos Genéticos, como algoritmos de búsqueda estocástica, basados en los mecanismos de selección natural, en los que sus cromosomas o estructuras de datos, que sufren la adaptación, son listas. La Programación Genética es un retoño de los Algoritmos Genéticos, en el que sus cromosomas, las estructuras de datos que sufren la adaptación, son árboles que representan programas. En otras palabras, son en sí mismos programas de computador. En Programación Genética, PG, se usan operadores genéticos especializados que generalizan la recombinación sexual y la mutación, para los programas de computador estructurados en árbol que están bajo adaptación. La PG, tiene las siguientes características:

- **Material genético no lineal y estructurado en árbol.**
  Aunque algunos Algoritmos Genéticos tienen material genético que no es lineal, el material genético lineal sigue siendo la regla en los Algoritmos Genéticos. Sin embargo, la PG opera sobre material genético no lineal, y explícitamente en estructuras de árbol.

- **Material genético de longitud variable.**
  La PG opera sobre material genético que puede variar de tamaño. Por razones prácticas, generalmente se implementan limitaciones en el crecimiento, pero normalmente permite crecimientos considerables a partir de la generación original que se produce aleatoriamente.

- **Material genético ejecutable.**
  La PG es la evolución directa de *programas* de computador. Así, en casi todos los casos el material genético que está evolucionando es en cierto sentido ejecutable. Aunque ejecutable no es el término más preciso, y sobre esto hay una serie de áreas grises. Generalmente las estructuras son interpretadas por algún interpretador, a veces en un lenguaje idéntico o muy parecido a un lenguaje de computación existente, a veces en un lenguaje diseñado para el problema a la mano. Sin embargo, en casi todos los casos hay un concepto de *ejecución* del material genético, con el objeto de ver directamente el comportamiento de la función deseada, a partir de la cual se obtiene la aptitud. En otras palabras, el genotipo es equivalente al fenotipo; en términos de sistemas complejos el programa es equivalente a la estructura, o en términos biológicos, el genotipo es el mismo fenotipo.

- **El Cruce preserva la sintaxis.**
  Aunque se han reportado muchos operadores de cruce para PG, en la mayoría de los casos están definidos de manera que preserva la corrección sintáctica del programa que es el

material genético, definido para cualquier lenguaje que se haya escogido para su representación.

4.2 Conceptos básicos de PG

El espacio de búsqueda en la PG es el espacio de todos los posibles programas de computador compuestos de funciones y terminales apropiados al dominio del problema. Las funciones pueden ser operaciones aritméticas, operaciones de programación, funciones matemáticas, funciones lógicas, o funciones específicas del dominio.

La PG es un intento de tratar con uno de los aspectos centrales en ciencias de la computación:

> *¿Cómo pueden aprender los computadores a solucionar problemas sin que se les programe explícitamente? En otras palabras, ¿cómo podemos hacer para que los computadores hagan lo que tienen que hacer, sin necesidad de decirles exactamente cómo lo deben hacer?*

Para aplicar la PG a un problema determinado, hay cinco pasos preparatorios importantes que se deben definir:

1. El conjunto de terminales,
2. El conjunto de funciones primitivas,
3. La medida de la aptitud,
4. Los parámetros para controlar la corrida, y
5. El método para designar un resultado y el criterio para terminar la ejecución del programa

Paso 1. Consiste en identificar el conjunto de terminales $\boldsymbol{T}$. Los terminales son las hojas de los árboles que corresponden a variables o a valores constantes, se pueden ver como las entradas al programa. El conjunto de terminales (junto con el conjunto de funciones) son los ingredientes a partir de los cuales la PG, trata de construir un programa de computador para solucionar total, o parcialmente, un problema.

Paso 2. Consiste en identificar el conjunto de funciones $\boldsymbol{F}$ que se va a utilizar, para generar la expresión matemática que trata de satisfacer la muestra dada finita de datos, o las propias del dominio. A cada función le corresponde una aridad (número de parámetros) y un tipo de datos. Ejemplo, la función seno requiere un solo parámetro, que es un número real. La función suma puede tener dos o más parámetros, los cuales pueden ser enteros o reales.

Cada programa de computador (árbol de análisis gramatical) es una composición de funciones del conjunto de funciones F y terminales del conjunto de terminales T.

Cada una de las funciones del conjunto de funciones debe ser capaz de aceptar, como sus argumentos, cualquier valor y tipo de dato que posiblemente, pueda retornar cualquier función del conjunto de funciones, y cualquier valor y tipo de dato que posiblemente, pueda tomar por cualquier terminal del conjunto de terminales. Esto es, el conjunto de funciones y el conjunto de terminales deben tener la propiedad de clausura.

En PG, se incuban genéticamente poblaciones de cientos, o más programas de computador. Esta incubación se hace usando el principio de supervivencia darwiniana, la reproducción de los más aptos, una operación de cruce genético apropiada para el apareamiento de programas de computador y una operación de mutación genética.

La PG comienza con una población inicial de programas de computador generados aleatoriamente compuestos de funciones y terminales apropiadas al dominio del problema. La creación de esta población inicial es, en efecto, una búsqueda aleatoria ciega en el espacio de búsqueda del dominio del problema, representado como programas de computador.

Paso 3. Cada programa de computador individual en la población, se mide en términos de que tan bien se comporta en el ambiente del problema particular. Esta medida se llama *medida de aptitud*. La naturaleza de la medida de aptitud varía con el problema.

Para muchos problemas, la aptitud de un programa se mide ponderando el error que se produce al ejecutar ese programa. El mejor programa de computador tendrá un error cercano a cero. En un problema de control óptimo, la aptitud de un programa de computador puede ser el tiempo (combustible o dinero) que consume el sistema para llevarlo al estado objetivo. Si se está tratando ejemplos de reconocimiento de patrones o de clasificación, la aptitud de un programa particular se puede medir por una combinación del número de instancias manejadas correctamente (cierto positivo, y falso negativo) y número de instancias manejadas incorrectamente (cierto negativo, y falso positivo).

Por otro lado, si se busca un buen generador de números aleatorios, la aptitud de un programa se puede medir por medio de entropía, satisfacción de la prueba de brecha, satisfacción de la prueba de corrida, o alguna combinación de estos factores. Para algunos problemas, puede ser apropiado usar una medida de aptitud multiobjetivo, que incorpore una combinación de factores tales como corrección o eficiencia.

En muchos casos, cada programa de la población se corre para varios *casos de aptitud*, de manera que su aptitud se mida como una suma o producto de una variedad de situaciones representativas diferentes. Estos casos de aptitud a veces representan un muestreo de valores diferentes de una variable independiente, o un muestreo de condiciones iniciales diferentes de un sistema. Por ejemplo, la aptitud de un programa individual se puede medir en términos de la suma del valor absoluto de las diferencias entre la salida producida por el programa y la respuesta correcta.

Los programas de computador de la población inicial, generación 0, generalmente tienen una aptitud excesivamente pobre. No obstante, algunos individuos de la población serán más aptos que otros. Estas diferencias en el comportamiento son las que se exploran y explotan posteriormente. La operación reproducción incluye la selección de un programa de la población actual de programas, con base en su aptitud, permitiéndole si es del caso, sobrevivir copiándolo en la nueva población.

La operación cruce se usa para crear nuevos hijos programas de computador, a partir de dos programas padres seleccionados. Los programas padres en PG, normalmente son de diferentes tamaños y formas. Los programas hijos se componen de subexpresiones (subárboles, subprogramas, subrutinas, bloques) de sus padres. Así, los programas hijos normalmente son de diferentes tamaños y formas que las de sus padres.

Intuitivamente, si dos programas de computador tienen alguna efectividad en solucionar un problema, entonces alguna de sus partes probablemente tenga algún mérito. La recombinación de partes escogidas aleatoriamente, de programas con alguna efectividad, a veces produce nuevos programas de computador que son aún más aptos para solucionar un problema determinado, que cualquiera de sus padres. En el siguiente numeral se justifica formalmente que en cada generación se encuentren mejores programas.

Paso 4. Después de que se efectúan las operaciones genéticas sobre la población actual, la población de hijos reemplaza la generación anterior, a cada individuo de la nueva población se le mide su aptitud, y el proceso se repite por muchas generaciones. En cada etapa de este proceso altamente paralelo, controlado localmente, el estado del proceso consistirá únicamente de la población actual de individuos. La fuerza que controla este proceso consiste solo en la aptitud observada de los individuos en su lucha con el ambiente problema. Normalmente, los parámetros que controlan la ejecución de PG son: el tamaño de la población; el número de generaciones; las probabilidades de aplicación de los operadores genéticos de cruce y mutación; y la estrategia de selección de los individuos para la nueva generación.

Paso 5. El mejor individuo que aparece en cualquier generación de una corrida, es decir el mejor hasta ahora, normalmente se designa como el resultado producido por la corrida de PG.

La variabilidad dinámica de los programas que se desarrollan para una solución, es otra característica de la PG. Generalmente es difícil y no es natural, tratar de especificar o limitar con anterioridad el tamaño y la forma de una solución eventual. Sin embargo, la especificación o limitación con anterioridad, del tamaño o la forma de la solución a un problema restringe la ventana a través de la cual el sistema ve el mundo y puede imposibilitar encontrar la solución de todo el problema.

Otra característica de la PG es la ausencia, o el papel relativamente mínimo, del preprocesamiento de entradas y el posprocesamiento de salidas. Las entradas, resultados intermedios, y salidas se expresan normalmente directamente en términos de la terminología natural del dominio del problema. El posprocesamiento de la salida del programa, si lo hay, se hace con una *envoltura* (interface de salida).

Finalmente, otra característica importante de la PG es que las estructuras que sufren la adaptación genética son estructuras activas. Es decir, como habíamos dicho, los cromosomas son el genotipo y también el fenotipo

4.3 Solución de problemas por PG

En resumen, la PG incuba programas de computador para la solución de problemas, ejecutando los siguientes pasos:

1. Generar una población inicial de composiciones aleatorias de funciones y terminales del problema (es decir, programas).

2. Ejecutar iterativamente los siguientes pasos hasta que se satisfaga el criterio de terminación:

- **a.** Ejecutar cada programa de la población y asignarle un valor de aptitud, de acuerdo con su comportamiento frente al problema.

b. Crear una nueva población de programas aplicando las siguientes dos operaciones primarias, a los programas escogidos, con una probabilidad basada en la aptitud.

i. Reproducir un programa existente copiándolo en la nueva población.

ii. Crear dos programas a partir de dos programas existentes, recombinando genéticamente partes escogidas de los dos programas en forma aleatoria, usando la operación cruce aplicada a un punto de cruce, escogido aleatoriamente dentro de cada programa.

iii. Crear un programa a partir de otro seleccionado aleatoriamente, cambiando aleatoriamente un gen (función o terminal). Es posible que se requiera un procedimiento de reparación para que se satisfaga la condición de clausura.

3. El programa identificado con la mayor aptitud (el mejor hasta la última generación), se designa como el resultado de la corrida de PG. Este resultado puede representar una solución (o una solución aproximada) al problema.

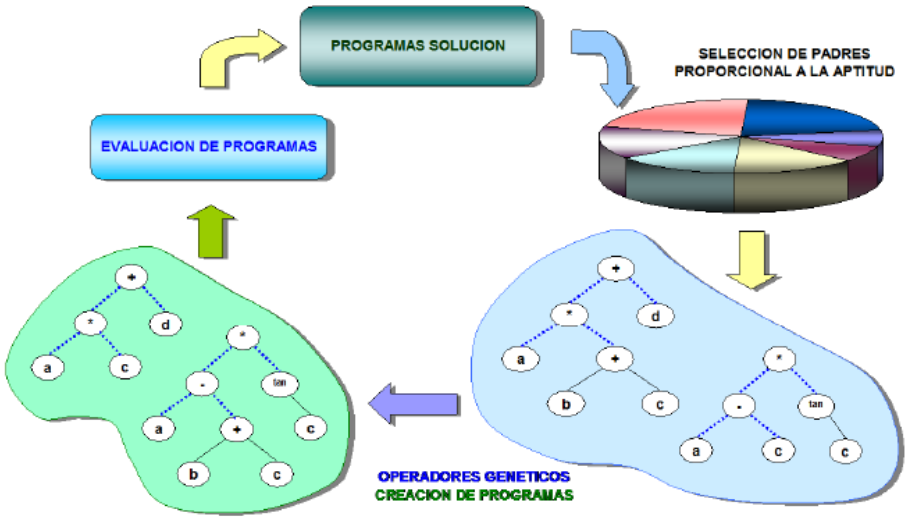


***Figura 4.1**. Estructura general de Programación Genética*

4.3.1 Ejemplo del operador cruce

El cruce genético (recombinación sexual) opera sobre dos programas padres y produce dos nuevos programas hijos consistentes de partes de cada padre.

Consideremos los siguientes programas:

$(a + b) * (c \uparrow e) + f$ y $(a * b) + (c + e) \uparrow f$

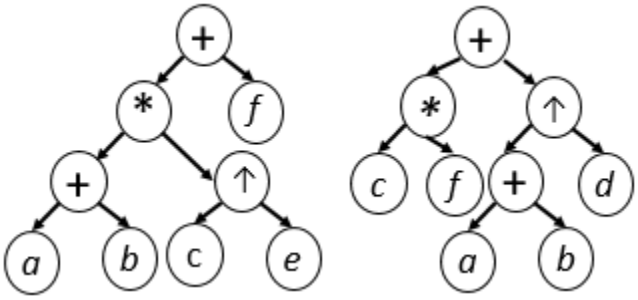
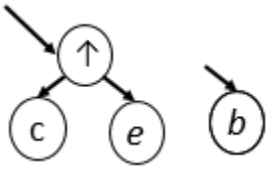


Figura 4.2. *Programas de computador padres*

En la figura 4.2 estos dos programas se presentan como árboles rotulados con ramas ordenadas. Los nodos, del árbol corresponden a funciones (es decir, operaciones) y las hojas, corresponden a terminales (es decir, datos de entrada). La operación cruce crea nuevos hijos, intercambiando subárboles (es decir, subrutinas, sub-procedimientos) entre los dos padres, figura 4.3.



$(c \uparrow e)$ $\quad$ b

Figura 4.3. *Dos fragmentos de cruce.*

Suponga que los nodos de ambos árboles se numeran en preorden (raíz, subárbol izquierdo, subárbol derecho). Suponga también, que se seleccionó aleatoriamente el nodo número 6, entre los 9 nodos del primer árbol, como punto de cruce para el primer padre y que se seleccionó aleatoriamente el nodo número 5, entre los 9 nodos del segundo árbol, como punto de cruce del segundo padre. Por lo tanto, los puntos de cruce son el ↑ para el primer padre y el operando b para el segundo padre.

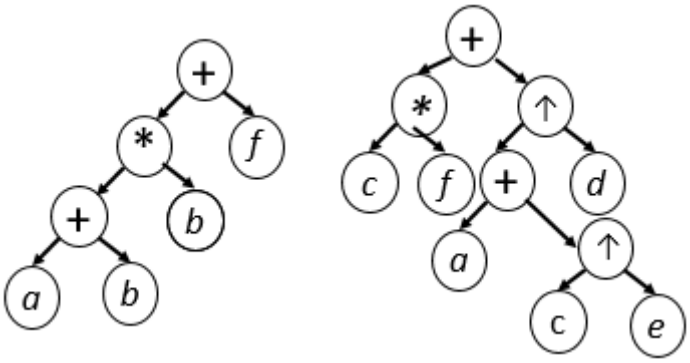


Figura 4.4*. Dos descendientes.*

Los hijos resultantes del cruce se muestran en la figura 4.4. De esta manera, el cruce crea nuevos programas usando partes de los programas padres existentes. Debido a que se intercambian subárboles completos, la operación cruce siempre produce programas sintáctica y semánticamente válidos, como descendientes sin considerar el punto de cruce. Debido a que los programas padres se han seleccionado para la operación cruce con una probabilidad basada en su aptitud, el cruce explora regiones del espacio de búsqueda con programas que contienen partes de programas promisorios.

Otro aspecto importante de observar es que mientras la profundidad de los árboles padres es 4, la profundidad del primer hijo es 4 y la profundidad del segundo hijo es 5. Esto lleva a que en la medida en que avanza el proceso genético va aumentando la profundidad de los árboles, lo que puede ocasionar en algunos casos la interrupción del programa por el manejo de pilas con muchos más

datos. En ese sentido a veces se limita la profundidad de los árboles, incluyendo este aspecto como una nueva restricción.

4.4 Estructuras de datos y operadores en PG

El mecanismo de la evolución, como se ha dicho, se basa en 3 operadores principales: selección, cruce y mutación. Estos operadores trabajan en conjunto con otros elementos; es así como en PG se debe contar con una población de expresiones simbólicas significativas, parámetros de control y una función de aptitud.

4.4.1 Mecanismo de codificación

En toda actividad que necesite una secuencia especial de pasos o tareas para llevar a cabo se tienen algoritmos y programas de computador. La PG busca representaciones simples y primitivas de tareas sencillas "que evolucionen" a formas más complejas y completas, que permitan resolver el problema de la mejor manera. La forma más común de representar estos programas es mediante funciones y parámetros. En matemáticas y lenguajes formales, una función es una representación simbólica que recibe varios valores (parámetros) a los que se les aplica varias operaciones, luego de lo cual entrega un valor o resultado, que es valor de la función.

Por ejemplo, la función $suma\ a\ +\ b$, o $suma\ (a, b)$ simplemente recibe dos parámetros (a y b) y después de aplicarles el operador " $+$ " devuelve el resultado. En teoría de compiladores y lenguajes formales, los algoritmos y las funciones, se pueden visualizar y se representan por medio de árboles.

Un árbol es una estructura de datos, que se define como "un conjunto finito de nodos T tales que: a) Hay un nodo raíz. B) Los otros nodos se particionan en M conjuntos $T_1, T_2, \ldots, T_M$. Los conjuntos $T_1, T_2, \ldots, T_M$ son árboles y se llaman subárboles de T. Cuando un árbol representa un programa, sus nodos internos representan funciones u operaciones y las hojas representan terminales o datos de entrada.

En PG los árboles se utilizan de un modo muy intuitivo y sencillo: todo padre es una función u operación, y sus hijos (los cuales pueden ser más de dos) son los parámetros u operandos. Claro está que un hijo puede ser a la vez el resultado de la operación sobre otros elementos, es decir, un hijo puede ser también una función. Si se tiene en cuenta que el resultado de aplicar una función sobre sus parámetros devuelve un valor, entonces todo hijo sigue de todos modos siendo un operando.

4.4.2 Conversión de árboles no binarios a binarios

Un caso particular de árboles es el de los *árboles binarios*, en los cuales el número de hijos de cualquier nodo está limitado a dos (subárbol izquierdo y subárbol derecho). En general, en ciencias de la computación se usa esta clase de árboles debido a la forma de su almacenamiento, que permite la relativa facilidad de implementación de sus operaciones y además son estructuras muy estudiadas, por esta razón se usa en PG.

En general algunos operadores pueden aceptar varios operandos, como es el caso de la " $+$ " o de la " $*$ "; o un solo operando como en el caso de la negación " $-$ "; o estrictamente dos operandos como en el caso de la división "/".

Supongamos que por alguna razón tenemos el siguiente árbol no binario y debemos pasarlo a binario, manteniendo la sintaxis:

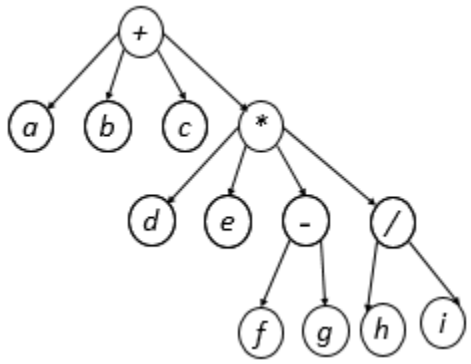


***Figura 4.5**. Árbol no binario que se va a convertir a binario.*

Para realizar la conversión se siguen las siguientes reglas:

1. Los operadores siempre quedan con solo dos operandos, uno un operando y el otro puede ser un operador.
2. Para esto se toma le primer operando de la izquierda como el subárbol izquierdo y como subárbol derecho se toma el operador raíz.
3. El siguiente operando es el subárbol izquierdo de esa raíz y el subárbol derecho nuevamente el operador raíz.
4. Se continua así hasta terminar los operandos.

Veamos paso a paso cómo se convierte a binario el árbol de la figura 4.5. Veamos el primer subárbol:

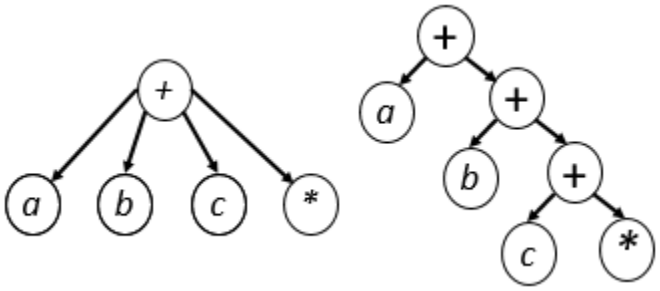


***Figura 4.6**. Primer subárbol convertido a binario no binario que se va a convertir a binario.*

La conversión completa se presenta en la figura 4.7.

4.5. Representación y recorrido de árboles binarios

Como se ha visto, el manejo de la estructura árbol es básico en PG, y por eso lo vamos a estudiar con algún detalle. Los árboles se pueden representar en memoria estática a través de estructuras matriz o a través de estructuras dinámicas. Aquí se trabajará el manejo dinámico; así comenzamos definiendo el nodo de un árbol:

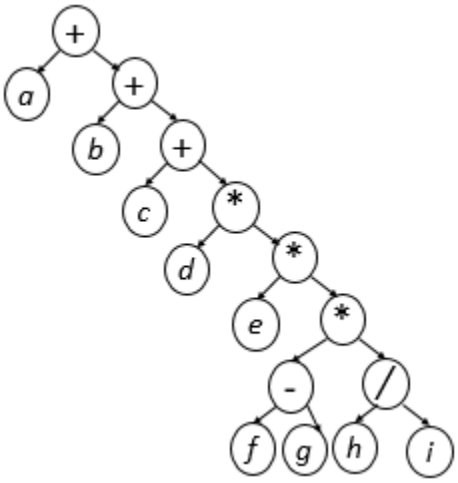


***Figura 4.7**. Árbol binario equivalente al árbol no binario de la figura 8.5.*

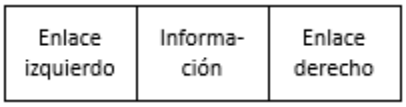


***Figura 4.8**. Nodo de un árbol binario*

Una operación muy importante es el recorrido de los árboles binarios. Recorrer un árbol significa visitar los nodos del árbol en forma sistemática, de esta manera se garantiza que se visitan todos los nodos una sola vez.

Hay tres métodos, recursivos, para hacer el recorrido de un árbol:

Preorden: Raíz, Subárbol Izquierdo, Subárbol Derecho.
Inorden: Subárbol Izquierdo, Raíz, Subárbol Derecho.
Postorden: Subárbol Izquierdo, Subárbol Derecho, Raíz.

Ejemplo. Se hacen los recorridos con respecto al árbol de la figura 4.9.

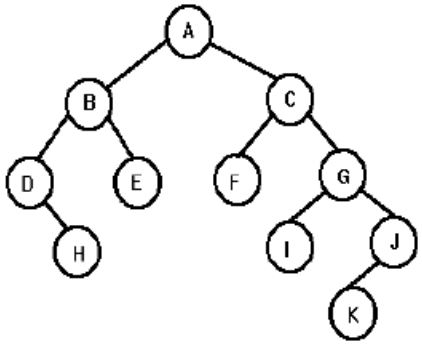


***Figura 4.9.** Árbol binario de ejemplo para hacer los recorridos.*

PREORDEN: ABDHECFGIJK.
INORDEN: DHBEAFCIGKJ.
POSORDEN: HDEBFIKJGCA.

Veamos el recorrido de un árbol en preorden. Se entra al árbol por el apuntador T que apunta a la raíz.

Un algoritmo muy útil en PG, es **Cuenta_Nodos,** que realiza el conteo de los nodos de un árbol, y es básicamente el algoritmo anterior, solo que en lugar de visitar los nodos los va contando.

4.6 Generación de árboles

A continuación, se presenta una forma de generación de árboles programa, teniendo como base el conjunto de funciones F y el conjunto de terminales T. Se supone un arreglo de Funciones, donde están las funciones y un arreglo de Terminales, donde están los terminales.

El primer algoritmo genera un árbol binario compuesto de raíz y dos hojas. Lo llamamos Genera_Subárbol. La raíz es una función escogida aleatoriamente del conjunto de Funciones, y sus dos hojas corresponden a dos terminales seleccionados aleatoriamente. Se supone que hay ***i*** funciones y ***j*** parámetros.

Un árbol se genera llamando iterativamente **Genera_Subárbol.** El subárbol T1 creado en **Genera_Subárbol,** reemplaza una hoja o terminal del árbol que se está generando. Este terminal se escoge aleatoriamente. El número de subárboles también puede ser aleatorio, sin embargo, no es necesario que el árbol tenga tanta profundidad. Se da como parámetro el número de nodos que se quiere, ***n***.

4.7 Cruce de árboles

El operador genético más importante en PG es el de cruce. En cada generación dos individuos de los más aptos combinan su información genética, formando dos nuevos individuos. Estos dos cromosomas son padres de los dos nuevos individuos formados al cruzarse entre sí. Por ejemplo, si tenemos las dos frases “Me gusta dormir tarde los domingos” y “Mi padre habla acerca del clima con mi novia” y las cruzamos (partiendo de algunos lugares aleatorios, denominado punto de cruce, como por ejemplo la cuarta y séptima palabras), podríamos obtener dos nuevas frases, con significados diferentes (se deja al lector el cruce como ejercicio). Lo importante aquí es notar cómo el cruce induce a la variedad en una población evolutiva, permitiendo con esto llegar a soluciones, en muchas ocasiones excelentes, en otras pocas realmente pobres.

En PG los elementos que se cruzan son programas. Una vez se tiene una población de árboles programa, se ejecutan y se mide la aptitud de cada uno, con respecto al problema que se quiere solucionar. Con base en esta medida de aptitud, se seleccionan los árboles programa padres para la siguiente generación. Hay varios métodos para hacerlo, que se pueden encontrar en la bibliografía de AG, por lo que aquí no nos vamos a ocupar de ellos.

Inicialmente, se escoge aleatoriamente una pareja de individuos para cruzar, y también aleatoriamente se escoge un punto en cada uno de ellos un punto de cruce. Se hace un traslado completo de subárboles de un árbol al otro y viceversa. Tomando la numeración de los nodos en preorden, se intercambian los subárboles. En la figura 4.10 vemos un ejemplo de cruce con dos programas muy sencillos (¿cuál sería el listado de código de estos programas?). Los nodos están enumerados en preorden.

Los pasos para realizar el cruce de dos árboles programas en PG son:

1. Seleccionar árboles a cruzar
2. Encontrar el número de nodos de cada árbol
3. Seleccionar el punto de cruce para cada árbol
4. Realizar el intercambio de subárboles por el punto de cruce escogido en cada árbol.

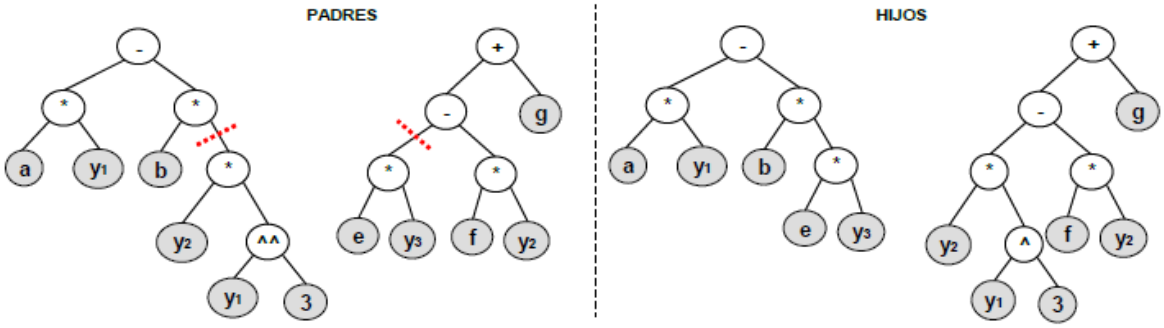


Figura 4.10. *Otro ejemplo del operador de cruce*

1. Seleccionar árboles a cruzar
Se escogen de la población de árboles seleccionados por su aptitud, aleatoriamente dos árboles, **T1** y **T2**.

2. Encontrar el número de nodos de cada árbol
Para cada árbol se cuenta el número de nodos que tiene con el algoritmo **Cuenta_Nodos (T,n).** Se corre para **T1** y **T2**, obteniendo respectivamente el número de nodos **n1** y **n2**.

3. Seleccionar el punto de cruce para cada árbol
Con base en n1, se escoge un punto de cruce **c1** para **T1** y con base en **n2**, se escoge un punto de cruce **c2** para **T2**. Se recorre cada árbol hasta el punto de cruce, encontrando el apuntador a ese nodo. Esto se hace con el algoritmo **Punto_Cruce (T, c, q)** que obtiene el apuntador **q** que direcciona el nodo **c** y la variable**. Ind_Enlace** nos indica porque subárbol de **q.**

4. Realizar el intercambio de subárboles
Una vez se tiene el apuntador de cada punto de cruce en su respectivo árbol, se intercambian los apuntadores, realizándose de esta manera el cruce.

Los algoritmos de mutación son equivalentes a los que se utilizan en el cruce (búsqueda del nodo) con la diferencia de que se debe cambiar el nodo objetivo por otro. Hay que tener la precaución de cambiar una función de ***i*** parámetros por una equivalente con los mismos parámetros y tipo, o una constante o variable por otra constante o variable.

4.8 Ejecución del programa árbol
Una vez se tiene la nueva población de árboles se deben correr para conocer cuál es su aptitud con respecto a los criterios que se han fijado. Para correr un programa, primero se convierte a notación

polaca y luego propiamente se ejecuta la expresión polaca. La expresión polaca se obtiene recorriendo el árbol en posorden.

Ejemplo, el recorrido en posorden del árbol de la figura 4.7 es el siguiente:

$abcdefg - hi/***+++$

Una vez se tiene la expresión en posorden se realiza la evaluación. Para esto se supone que la expresión está en el vector X, con n elementos.

Los akgoritmos se presentan en el Notebook de Juoyter.

4.9 Ejemplo de PG: Regresión Simbólica

La regresión simbólica consiste en obtener una función algebraica que ajuste en gran medida un conjunto de datos experimentales. Similar a la muy conocida regresión por mínimos cuadrados, la diferencia es que en la regresión simbólica no se conoce el formato de la función a ajustar, por lo tanto, se debe obtener la función y los mejores coeficientes que ajusten los datos experimentales.

De los datos experimentales se pueden reconocer las variables dependientes e independientes del problema, en el caso de la figura 4.11, existe una variable dependiente y y dos variables independientes x_1, x_2; entonces el conjunto de terminales del problema PG serán las variables independientes y los coeficientes de la función a ajustar. El conjunto de funciones se puede limitar a las funciones elementales: $\{+, *, /, -\}$. Nótese que el árbol como programa se ejecuta cuando se evalúa el modelo algebraico con los terminales como entradas.

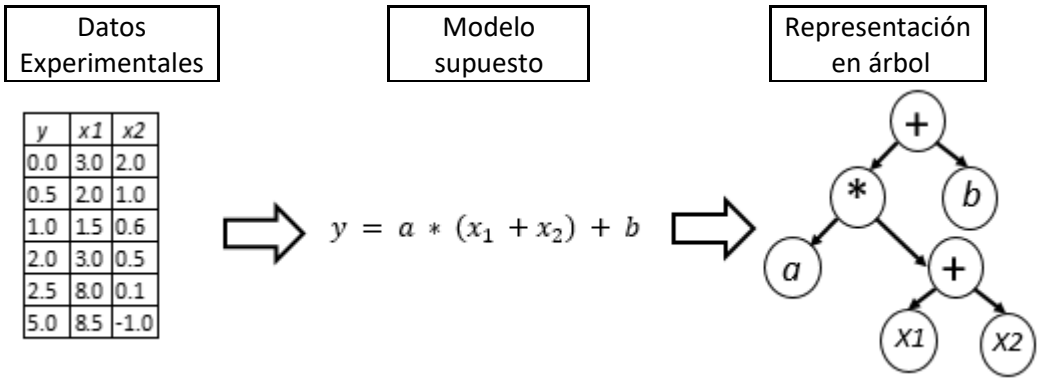


Figura 4.11. *Representación en árbol de un modelo supuesto, para un conjunto de datos experimentales.*

La función de aptitud es entonces un indicativo que permite premiar a los individuos, árboles, que se ajustan mejor a los datos experimentales. Para encontrar una función de aptitud útil se puede utilizar el error absoluto, la diferencia de valores evaluados con el modelo supuesto y los datos experimentales. El error absoluto es menor para los árboles más aptos, de manera que el error es inversamente proporcional a la aptitud, por lo que definimos la función de aptitud f_{apt},

$$f_{apt} = \frac{1}{\sum_j |y_j - yc_{i,j}|}$$

Donde los y_j son los valores de la tabla, en este caso $j = 1, \ldots, 6$. Y los $yc_{i,j}$ son los valores calculados por todos los arboles i de la población, para la posición j de la tabla. Así que f_{apt} indica que la aptitud del $i - ésimo$ árbol es igual al inverso de la suma de errores absolutos (donde es el valor del $j - ésimo$ dato experimental evaluado en las variables independientes; mientras que, es el valor evaluado en el $i - ésimo$ árbol; por lo tanto, si el error es pequeño la aptitud será muy grande. De manera que, si el error es cero, el valor de la función de aptitud tenderá a infinito; esto se evita en la práctica colocando en el denominador un número relativamente pequeño para fijar un límite superior:

$$f_{apt} = \frac{1}{(\,0.1 + \sum_j |y_j - yc_{i,j}|)}$$

Entonces por la ecuación anterior tendrá como máxima aptitud f_{apt}, el valor de 10, para un error cero.

Supongamos que se ha corrido un ejemplo para el mismo programa, árbol, de la figura 8.11, para valores obtenidos de $a = 1.5$ y $b = -1.0$, se han encontrado los valores yc respectivos. La diferencia en valor absoluto de $|y_j - yc_{i,j}|$ es el error. La f_{apt} se encuentra haciendo primero la sumatoria del error para cada valor de la tabla, 6.5, luego se suma el 0.1 y se saca el valor inverso para obtener $f_{apt} = 0.1515$

yc	*x1*	*x2*	Error
1.5	3.0	2.0	1.5
0.5	2.0	1.0	3E-11
0.05	1.5	0.6	0.95
0.75	3.0	0.5	1.25
3.05	8.0	0.1	0.55
2.75	8.5	-1.0	2.25
Error total			6.5
Aptitud 0.1515			

Figura 4.11. *Representación en un árbol del modelo supuesto, para un conjunto de datos experimentales.*

Los parámetros de control para este ejemplo de PG se fijaron con los valores recomendados [7], probabilidad de cruce = 0.7; probabilidad de mutación 0.01; tamaño de la población= 100, generaciones = 500, Criterio de selección = elitismo [4]. Por lo tanto, el sistema de PG se detiene cuando se cumple el número de generaciones y la solución será el árbol con mejor aptitud.

En el transcurso de las generaciones y gracias al operador de cruce los árboles empiezan a crecer especialmente en profundidad, generando árboles muy profundos que aportan muy poco al

incremento de la aptitud. Así que es conveniente fijar un tamaño máximo para el tamaño de los árboles de la población, o incluso un cruce que haga esta limitación.

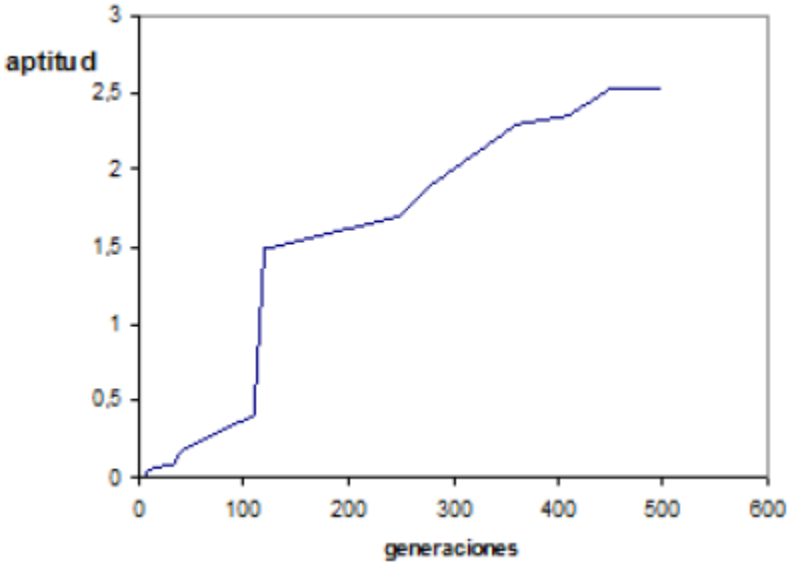


***Figura 4.12**. Historia de la aptitud para una regresión simbólica*

En la figura 4.12, se puede observar cómo varía la aptitud, del mejor individuo en cada generación, al final se obtiene una aptitud de 2.53456 con la siguiente expresión como solución:

$$y = a * x_1 - b * x_2 * x_2 + c * (d * x_1 + e * x_2)/(x_1 - x_2 + f)$$

4.10. Múltiples expresiones en un solo cromosoma

Hay una serie de técnicas de PG, entre las que está la de múltiples expresiones ne un solo cromosoma. La idea es un cambio en la estructura de los cromosomas en el sentido de tener varias expresiones en el cromosoma, MEP. Lo que significa que podemos hacer una exploración mucho más profunda del espacio de soluciones programa.

https://www.mepx.org/

4.11 Aplicaciones

- Control óptimo https://www.youtube.com/watch?v=K2Hl7m2Ty_4

- Planeación de trayectorias
- Diseño de circuitos lógicos
- Planeación de trayectorias
- Estrategia de juegos
- Programación de máquinas
- Machine Learning
- Programación de computadores
- Reingeniería

4.12. Ejercicios

De los ejercicios siguientes, escoja 2 para realizar.

1. Estudie el MEPX, https://www.mepx.org/. Con base en una tabla histórica de 60 datos del valor de una acción determinada o de TRM, o del índice de precios al consumidor, encontrar una fórmula que se ajuste dicha curva usando MEPX. Con base en esa fórmula pronostique, el valor de esa acción o índice, para los siguientes 5 días.
2. Estudie en https://gplearn.readthedocs.io/en/stable/intro.html#classification, la librería gplearn de Python; y utilícela para solucionar el ejercicio anterior u otro problema que le interese.
3. Suponga que desea utilizar Programación Genética para encontrar el circuito lógico de un codificador de 7 segmentos, a partir de un byte con un número decimal que debe mostrarlo en un LED de 7 segmentos. Describa el conjunto de terminales, el conjunto de funciones y la función de aptitud. Use una librería de Python.
4. Suponga que tiene un robot que le entrega galletas al grupo de ingenieros de diseño de robots. Programe por PG el recorrido del robot, teniendo en cuenta que cada vez que un ingeniero recibe una galleta gana puntos. Los ingenieros están distribuidos en una sala cuadrada. Defina, conjunto de terminales, conjunto de funciones y función de aptitud.
5. ¿Puede utilizarse la PG para hacer Ingeniería Inversa? ¿Justifique, con todo el detalle, su respuesta?

4.11 Bibliografía

6. [1] Martínez J y Rojas S. *Introducción a la informática Evolutiva.* Bogotá, Colombia: Universidad Nacional de Colombia. Primera Edición. 1999.
7. [2] Langdon W.B y Poli R, *Foundations of Genetic Programming* . Berlin: Springer-Verlag, 2002.
8. [3] Michalewicz Z, *Genetic Algorithms+Data Structures=Evolution Programs* . Berlin: Springer-Verlag, 1994.
9. [4] Goldberg D, *genetic Algorithms in search, optimization and Machine learning*. Massachusetts: Addison-Wesley,Reading, 1989.
10. [5] Gen M y Cheng R, *Genetic Algorithms and engineering design*. New York: John Wiley & Sons, 1997.
11. [6] McKay B., Willis M y Barton G, "Steady-state modelling of chemical process systems using genetic programming," *Comp & Chem Eng.*, vol 21, no 9, pp. 981-996, 1997.
12. [7] Koza J, *Genetic Programming, on the programming of computers by means of natural selection*. The MIT Press, 1992
13. [8] Poli R y Langdon W.B. "Schema theory for genetic programming with one-point crossover and point mutation". *Evolutionary Computation*, 6(3): 231-252, 1998.
14. [9] Martínez J. Estructuras de Información. Bogotá, Colombia: Universidad Nacional de Colombia. Primera Edición. 2000.
15. [10] Torres Juan Esteban, Martínez José Jesús, *Departamento de Ingeniería de Sistemas e Industrial, Universidad Nacional de Colombia, Bogotá,* 2004.