4. PROGRAMACION GENETICA CONCEPTOS BASICOS

José J. Martínez P.
Jmcursosingenieria111@gmail.com
Febrero 26

4.1 Programación Genética

Hablamos de los Algoritmos Genéticos, como algoritmos de búsqueda estocástica, que se basan en los mecanismos de la selección natural, en los que sus cromosomas son listas. La Programación Genética, PG, es un retoño de los Algoritmos Genéticos, en el que sus cromosomas, que sufren la adaptación, son árboles que representan programas. Es decir, 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 forma de árbol que están bajo adaptación. La PG, tiene las siguientes características:

- **Material genético no lineal y estructurado en árbol.**
  La PG opera sobre material genético no lineal, y explícitamente en estructuras de árbol.

- **Material genético de longitud variable.**
  El material genético puede variar de tamaño. Por razones prácticas, se limita el crecimiento de los árboles, aunque se permiten crecimientos 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 es en cierto sentido ejecutable. Generalmente las estructuras las interpreta algún interpretador, en un lenguaje idéntico o muy parecido a un lenguaje de computación, o a veces en un lenguaje diseñado para el problema a resolver. En casi todos los casos hay un concepto de *ejecución* del material genético, con el objeto de ver el comportamiento de la función deseada, para obtener la aptitud. En otras palabras, el genotipo es equivalente al fenotipo.

- **El Cruce preserva la sintaxis.**
  El cruce 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?*

Es importante anotar los desarrollos de IA Generativa en la producción de código como de Chat-GPT3, aiXcoder, Mintlify, Copilot, que generan código con un enfoque completamente diferente al de la PG. En sus últimas aplicaciones la PG se utiliza en la producción de lenguajes funcionales. Volveremos luego con este tema.

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

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

Paso 1. Debe haber claridad con respecto con respecto al problema que se pretende solucionar por PG, como puede ser el análisis de datos a través de regresiones, clasificación y series de tiempo; entre otros

Paso 2. Los terminales son las hojas de los árboles que corresponden a variables o constantes, son las entradas al programa.

Paso 3. 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 expresiones 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 T.

Cada una de las funciones de F debe aceptar, como argumentos, cualquier valor y tipo de dato, que pueda retornar otra función de F, y cualquier valor que pueda tomar de cualquier terminal T. Esto es, el conjunto de funciones y el conjunto de terminales deben tener la propiedad de clausura.

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

En muchos problemas la aptitud de un programa se mide ponderando el error al ejecutarlo. El mejor programa de computador tendrá un error cercano a cero. En control óptimo, la aptitud de un programa puede ser el tiempo, combustible o dinero que consume el sistema. En reconocimiento de patrones o clasificación, la aptitud de un programa se mide por una combinación del número de instancias manejadas correctamente y número de instancias manejadas incorrectamente.

En muchos casos, cada programa de la población se corre para varios *casos de aptitud*, de manera que su aptitud se mide como una suma o producto de varias situaciones representativas. Por ejemplo, la aptitud de un programa 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 la población inicial, generación 0, tienen una aptitud muy pobre. No obstante, algunos individuos serán más aptos que otros. 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, a partir de dos programas padres seleccionados. Los programas padres, normalmente son de diferentes tamaños y formas. Los programas hijos se componen de subexpresiones, subárboles, subprogramas, de sus padres. Así, los programas hijos normalmente son de diferentes tamaños y formas que las de sus padres.

Intuitivamente, si dos programas 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 que son aún más aptos para solucionar un problema determinado, que cualquiera de sus padres.

Paso 5. Después de efectuar 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, el estado del proceso consiste 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. Los parámetros que controlan la ejecución de PG son: el tamaño de la población; el número de generaciones; la probabilidad de aplicación del operador de mutación; y la estrategia de selección de los individuos para la nueva generación.

Paso 6. 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.

Otra característica de la PG es la ausencia de pre-procesamiento de las entradas y el pos-procesamiento de salidas, ya que el genotipo y el fenotipo son lo mismo. Las entradas, resultados intermedios, y salidas se expresan normalmente directamente en términos de la terminología del dominio del problema. El pos-procesamiento de la salida, si lo hay, se hace con una interface de salida.

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.

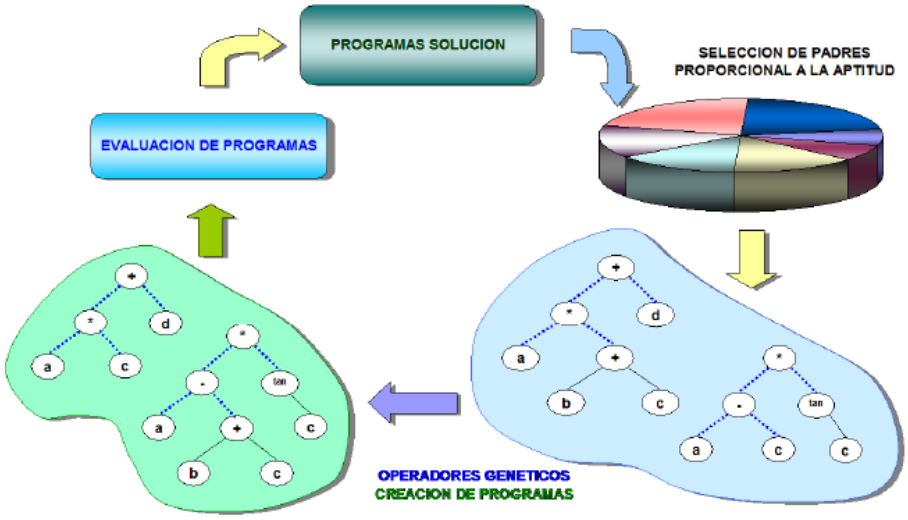


***Figura 4.1**. Estructura general de Programación Genética*

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 $(c * f) + ((a + b) \uparrow d)$

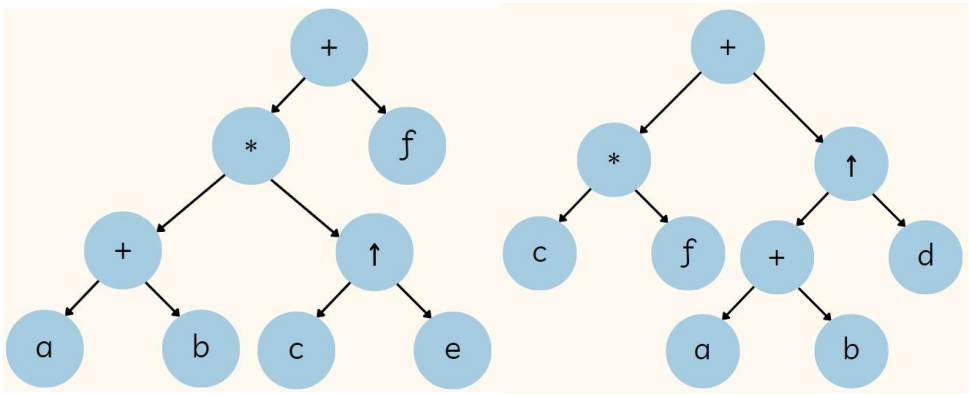
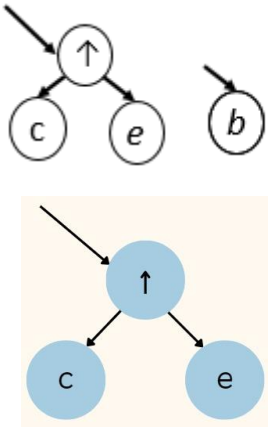


Figura 4.2. *Programas de computador padres*

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.



$(c \uparrow e)$ 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.

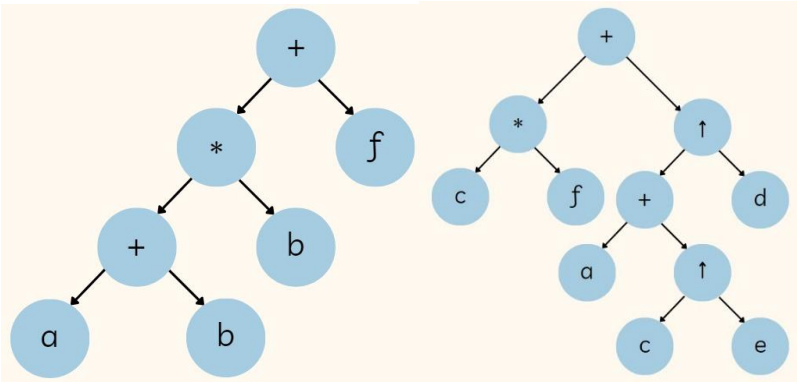


Figura 4.4*. Dos descendientes.*

Los hijos resultantes del cruce se muestran en la figura 4.4, $(a + b) * b + f$ y $(c * f) + (a + (c \uparrow e) \uparrow d)$

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. 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 al avanzar el proceso evolutivo aumenta la profundidad de los árboles, lo que puede llevar a la interrupción del programa por el manejo de sus pilas.

4.4 Estructuras de datos y operadores en PG

4.4.1 Mecanismo de codificación

En toda actividad que necesite la ejecución de una secuencia especial de pasos o tareas se tienen algoritmos y programas de computador. La PG busca representaciones simples de tareas sencillas "que evolucionen" a formas más complejas, que permitan resolver el problema de la mejor manera. La forma más común de representar estos programas es mediante funciones y parámetros. 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 y las hojas representan terminales o datos de entrada.

4.5. Representación y recorrido de árboles binarios

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 tipo matriz o a través

de estructuras dinámicas. Aquí se trabajará el manejo dinámico; así comenzamos definiendo el nodo de un árbol:

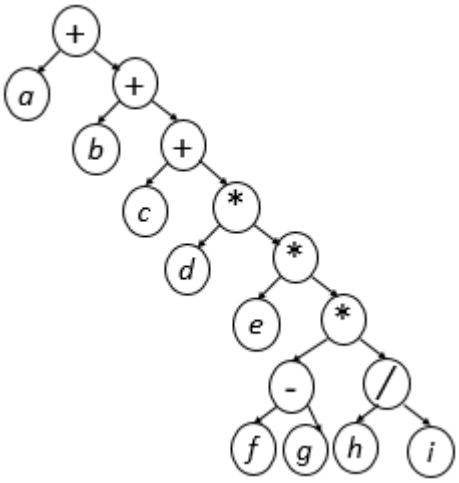


***Figura 4.7**. Árbol binario equivalente al árbol no binario de la figura 8.5.*

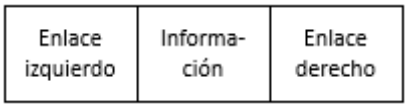


***Figura 4.8**. Nodo de un árbol binario*

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.

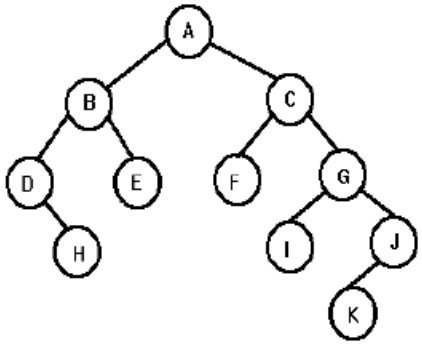


***Figura 4.9.** Árbol binario de ejemplo para hacer los recorridos.*

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

Es el operador más importante en PG es el cruce. En cada generación dos individuos de los más aptos combinan su información genética, formando dos nuevos individuos. 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 (con un punto de cruce, como por ejemplo la cuarta y sexta palabras), se obtienen dos nuevas frases, con significados diferentes. Lo importante aquí es notar cómo el cruce induce a la variedad en una población evolutiva, permitiendo llegar a soluciones.

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 una pareja de individuos para cruzar, y 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.

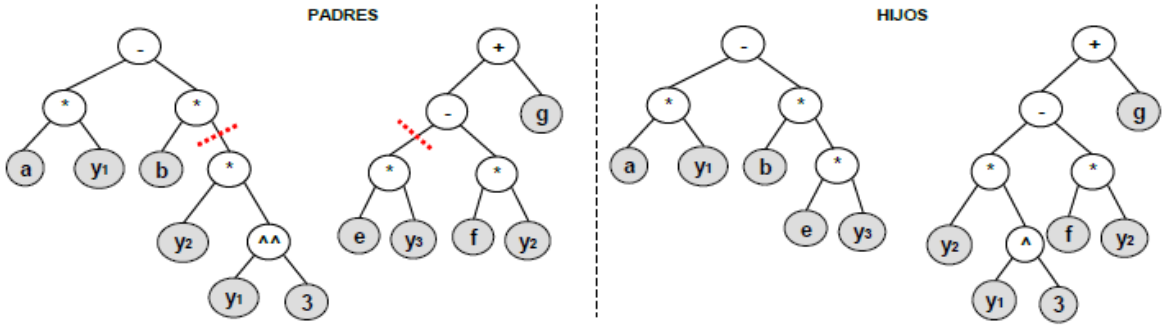


Figura 4.10. *Otro ejemplo del operador de cruce*

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 algoritmos se presentan en el Notebook de Jupyter.

4.9 Aplicaciones

Síntesis general de programas

Es un área cada vez más importante de aplicación de la PG. El CBGP (Code Building Genetic Programming) permite hacer síntesis general de programas que soporta la evolución de programas que pueden usar tipos de datos arbitrarios y polimorfismo. Entre estos se encuentran varias herramientas CAD como son: Clojure, es un lenguaje dinámico de propósito general, dialecto de LISP; OpenSCAD, un compilador 3D; QCad, que es una CAD para 2D; FreeCAD, es un modelador paramétrico 3D; e InkScape, que es un editor de gráficos vectoriales.

Programación de FPGAs

El hardware evolutivo es un concepto relativamente nuevo y moderno en el diseño de circuitos, que se basa en estructuras reconfigurables. Los sistemas evolutivos pueden cambiar dinámicamente su función y sus parámetros. En este sentido los FPGAs son muy útiles.

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.

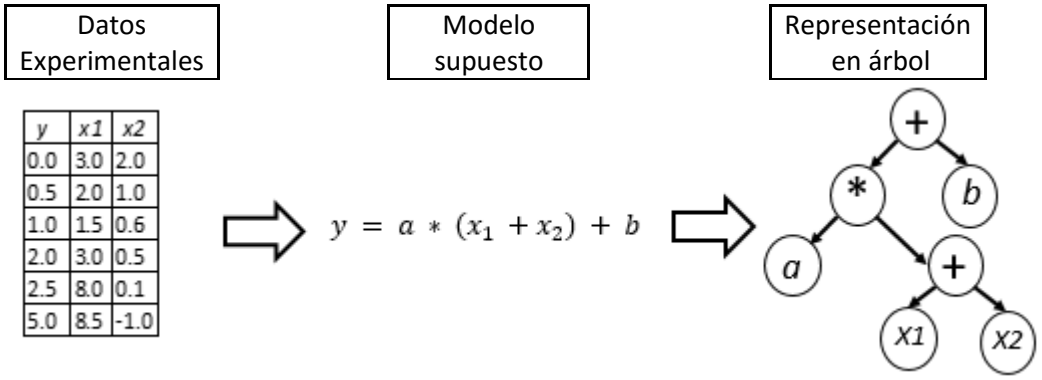


Figura 4.11. *Representación en árbol de un modelo supuesto, para un conjunto de datos experimentales.*

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},

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

Donde los y_j son los valores de la tabla, en este caso $j = 1, \dots, 6$. 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:

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

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 $\left| y_j - yc_{i,j} \right|$ 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.

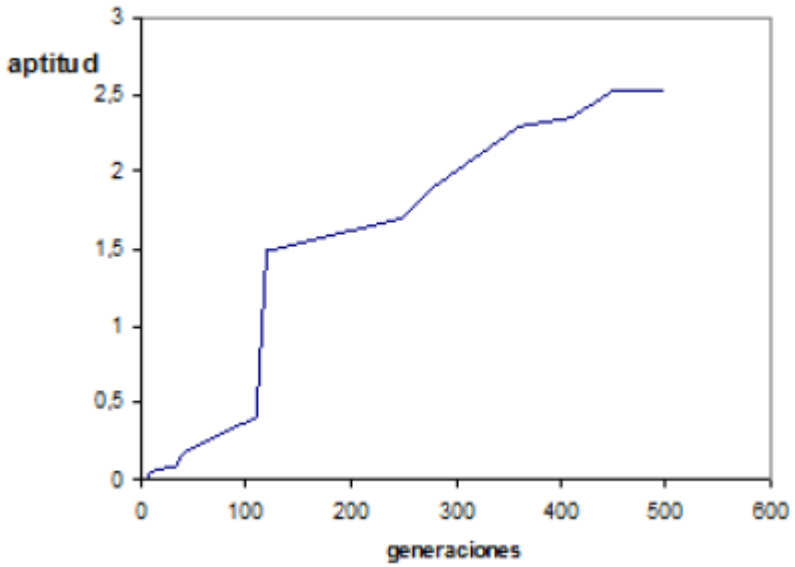


***Figura 4.12**. Historia de la aptitud para una regresión simbólica*

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

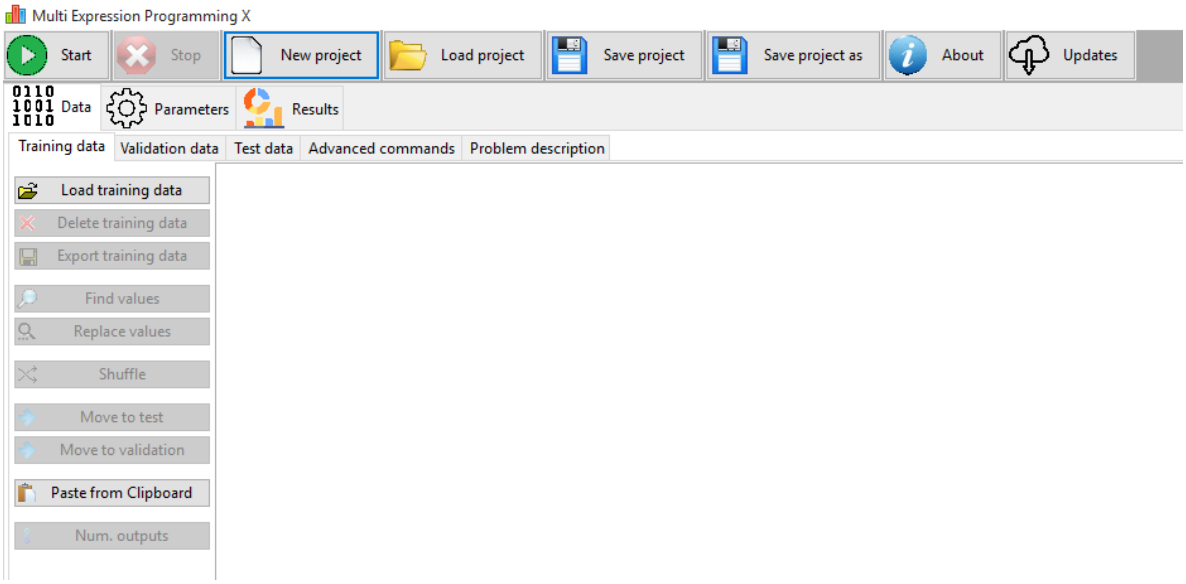
Hay una serie de técnicas de PG, entre las que está la de tener múltiples expresiones en un solo cromosoma. La idea es un cambio en la estructura de los cromosomas en el sentido de tener varias expresiones en el cromosoma, MEP. Lo que significa que podemos hacer una exploración mucho más profunda del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

Representación

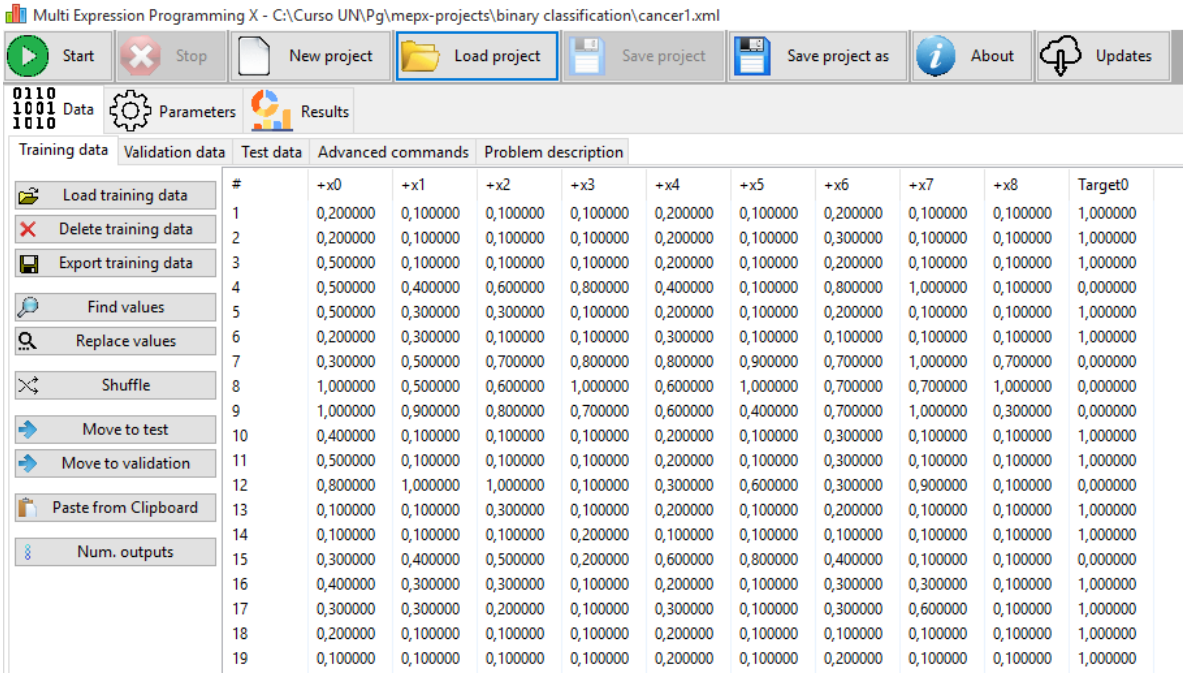
La representación se hace con base en listas que son representaciones de árboles, inspirada en el código three-address, que es un código intermedio que usan los compiladores optimizados para ayudar en la implementación de transformaciones para mejorar el código. Así que cada cromosoma tiene un número constante de genes, donde cada gen corresponde a una instrucción o a una variable, una terminal. Los genes que codifican una función incluyen un apuntador a los argumentos de la función.

MEPX

Un paquete muy interesante y poderoso es el MEPX, con el cual se pueden trabajar una serie de aplicaciones y que se puede descargar de https://www.mepx.org/. Incluye una serie de ejemplos con los que se puede experimentar en clasificación y regresión. La siguiente es la entrada:

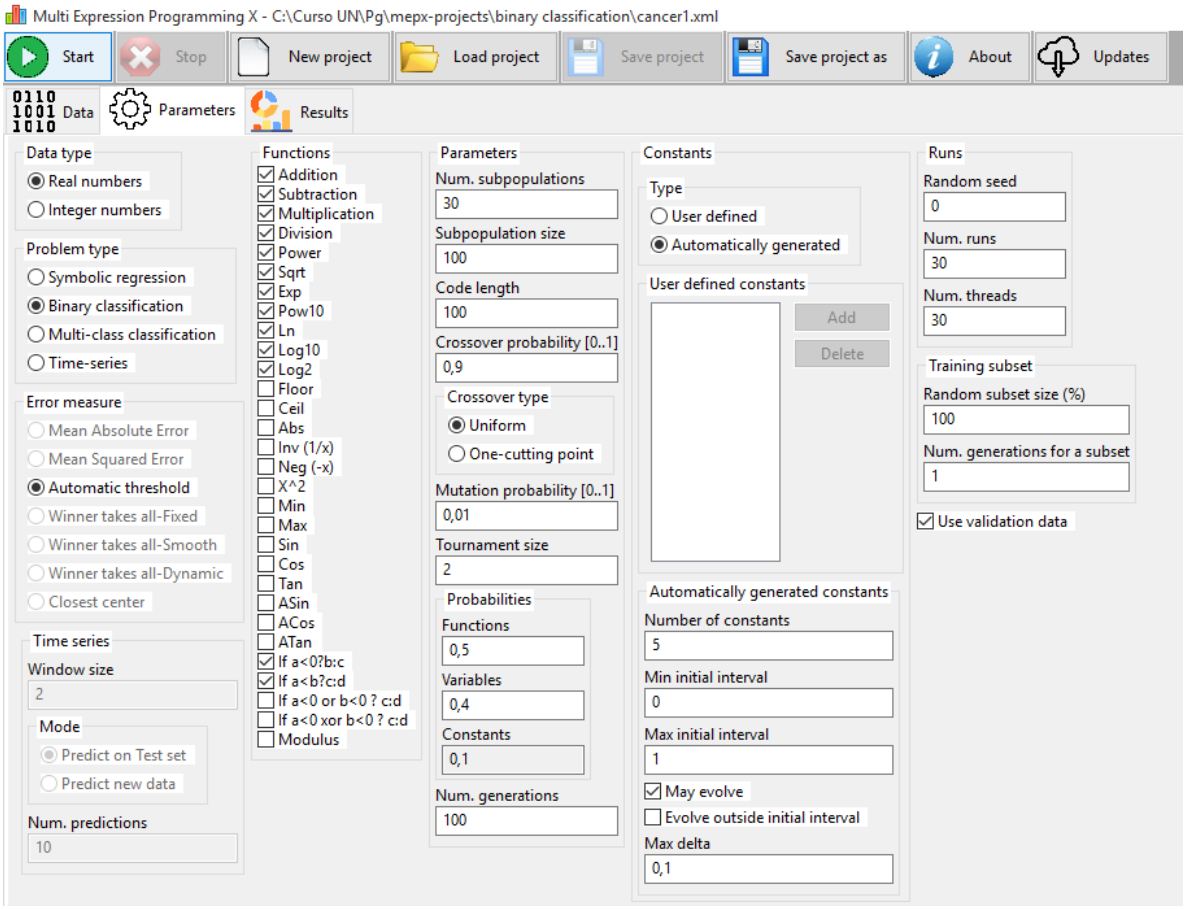


Luego se escoge uno de los ejemplos, por ejemplo "Cáncer 1" que incluye los datos relacionados con las variables, “features”, utilizados para determinar si una persona tiene o no cáncer.



#	+x0	+x1	+x2	+x3	+x4	+x5	+x6	+x7	+x8	Target0
1	0,200000	0,100000	0,100000	0,100000	0,200000	0,100000	0,200000	0,100000	0,100000	1,000000
2	0,200000	0,100000	0,100000	0,100000	0,200000	0,100000	0,300000	0,100000	0,100000	1,000000
3	0,500000	0,100000	0,100000	0,100000	0,200000	0,100000	0,200000	0,100000	0,100000	1,000000
4	0,500000	0,400000	0,600000	0,800000	0,400000	0,100000	0,800000	1,000000	0,100000	0,000000
5	0,500000	0,300000	0,300000	0,100000	0,200000	0,100000	0,200000	0,100000	0,100000	1,000000
6	0,200000	0,300000	0,100000	0,100000	0,300000	0,100000	0,100000	0,100000	0,100000	1,000000
7	0,300000	0,500000	0,700000	0,800000	0,800000	0,900000	0,700000	1,000000	0,700000	0,000000
8	1,000000	0,500000	0,600000	1,000000	0,600000	1,000000	0,700000	0,700000	1,000000	0,000000
9	1,000000	0,900000	0,800000	0,700000	0,600000	0,400000	0,700000	1,000000	0,300000	0,000000
10	0,400000	0,100000	0,100000	0,100000	0,200000	0,100000	0,300000	0,100000	0,100000	1,000000
11	0,500000	0,100000	0,100000	0,100000	0,200000	0,100000	0,300000	0,100000	0,100000	1,000000
12	0,800000	1,000000	1,000000	0,100000	0,300000	0,600000	0,300000	0,900000	0,100000	0,000000
13	0,100000	0,100000	0,300000	0,100000	0,200000	0,100000	0,200000	0,100000	0,100000	1,000000
14	0,100000	0,100000	0,100000	0,200000	0,100000	0,100000	0,100000	0,100000	0,100000	1,000000
15	0,300000	0,400000	0,500000	0,200000	0,600000	0,800000	0,400000	0,100000	0,100000	0,000000
16	0,400000	0,300000	0,300000	0,100000	0,200000	0,100000	0,300000	0,300000	0,100000	1,000000
17	0,300000	0,300000	0,200000	0,100000	0,300000	0,100000	0,300000	0,600000	0,100000	1,000000
18	0,200000	0,100000	0,100000	0,100000	0,200000	0,100000	0,100000	0,100000	0,100000	1,000000
19	0,100000	0,100000	0,100000	0,100000	0,200000	0,100000	0,200000	0,100000	0,100000	1,000000

Presenta la operaciones que se van a utilizar seleccionadas de un conjunto de todas las posibles operaciones que se pueden utilizar.



Se inicia el entrenamiento del modelo utilizando los parámetros previamente definidos. El m odelo se ajusta iterativamente a los datos de entrenamiento, y se calcula un error con respec to a la validación para evaluar su rendimiento.

MEPX proporciona métricas de evaluación del rendimiento del modelo, como el error prome dio en comparación con los datos de validación. Esto nos permite determinar qué tan bien se desempeña el modelo en la tarea de clasificar la presencia o ausencia de cáncer.

Después del entrenamiento exitoso, MEPX genera una salida que proporciona detalles sobre el número de generaciones y corridas realizadas. También genera automáticamente un códig o en Python (cáncer.py en este caso) que se puede utilizar para aplicar el modelo entrenado para llevarlo a otros sistemas o plataformas.

4.11 Relación entre PG e IA Generativa para la generación automática de programas.

La generación automática de programas ha sido un campo de estudio e investigación en IA. El enfoque original era utilizar la PG. Sin embargo, con la aparición de la IA Generativa, con modelos de lenguaje especializados, se han superado una serie de barrera que todavía se mantenían. Veamos la siguiente tabla comparativa.

Aspecto	Programación Genética (GP)	Modelos Generativos de IA (LLMs para código)

Inspiración	Evolución biológica	Aprendizaje estadístico profundo, modelos de lenguaje
Generación de código	Evolución de árboles	Predicción de código basado en contexto
Representación interna	Árboles sintácticos o estructuras de instrucciones	Texto plano (tokens de código fuente)
Mecanismo de mejora	Selección natural basada en aptitud	Aprendizaje supervisado
Exploración del espacio	Estocástica y evolutiva	Probabilística, basada en patrones del corpus de entrenamiento
Objetivo	Encontrar soluciones que optimicen una función objetivo	Generar código correcto y relevante según el prompt
Control del proceso	Alto: se define la función de aptitud y operadores genéticos	Bajo: se define el prompt, pero el modelo decide la solución
Personalización	Alta: se puede diseñar la aptitud específica para el problema	Moderada: se puede ajustar vía prompt o fine-tuning
Creatividad / Sorpresa	Alta: puede generar soluciones no convencionales	Alta: puede combinar patrones en formas nuevas
Limitaciones	Requiere muchas evaluaciones y puede ser lento	Puede "alucinar" o generar errores sutiles

Similitudes y Diferencias

Similitudes
En ambos enfoques se busca generar automáticamente programas de computador, código, sin programación manual directa. También se requiere que los programas que generan sean validados por humanos para determinar su calidad y su posible utilización. Los enfoques de ambas permiten generar soluciones creativas no triviales. Mientras en la PG, la generación de programas

Diferencias
La PG se basa en un proceso evolutivo lento, mientras que con los modelos generativos, la predicción es inmediata. En la PG, se tiene una gran población de posibles soluciones que se trabajan en paralelo, mientras que en el enfoque de IA generativa, se genera de a una solución. Mientras que en la PG la generación de programas es altamente controlable e interpretable, en la IA Generativa es opaca, aunque muy rápida.

4.12 Aplicaciones

- Planeación de trayectorias
- Diseño de circuitos lógicos
- Estrategia de juegos
- Programación de máquinas
- Machine Learning
- Programación de computadores
- Reingeniería
- Control óptimo https://www.youtube.com/watch?v=K2Hl7m2Ty_4

4.12. Ejercicios

De los ejercicios 1 y 2, escoja 1 para realizar; y hacer 3 o 4.

1. Descargue MEPX, https://www.mepx.org/, estúdielo y corra uno de los ejemplos que trae.
2. Suponga que desea utilizar Programación Genética para encontrar el diseño de un circuito lógico, tome como, ejemplo el codificador de 7 segmentos. Describa el conjunto de terminales, el conjunto de funciones y la función de aptitud. Use una librería de Python.
3. 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.
4. Vea el video, https://www.youtube.com/watch?v=6KNuJn6dVy4. Analícelo y haga un ejemplo de control aplicando PG.

4.11 Bibliografía

[1] Gen M y Cheng R, *Genetic Algorithms and engineering design*. New York: John Wiley & Sons, 1997.

[3] Koza J, *Genetic Programming, on the programming of computers by means of natural selection*. The MIT Press, 1992

[4] Langdon W.B y Poli R, *Foundations of Genetic Programming* . Berlin: Springer-Verlag, 2002.

[5] Martínez J. Estructuras de Información. Bogotá, Colombia: Universidad Nacional de Colombia. Primera Edición. 2000.

[6] Martínez J y Rojas S. *Introducción a la informática Evolutiva.* Bogotá, Colombia: Universidad Nacional de Colombia. Primera Edición. 1999.

[7] 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.

[8] Michalewicz Z, *Genetic Algorithms+Data Structures=Evolution Programs* . Berlin: Springer-Verlag, 1994.

[9] 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.

[10] Torres Juan Esteban, Martínez José Jesús, *Departamento de Ingeniería de Sistemas e Industrial, Universidad Nacional de Colombia, Bogotá,* 2004.

[11] Su Nguyen1, Yi Mei1, Mengjie Zhang1, Genetic programming for production scheduling: a survey with a unified framework, This article is published with open access at Springerlink.com, 2017.