51
2ej.

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA DE ESTUDIOS PROFESIONALES
A C A T L A N

COMPUTACION EVOLUTIVA

T E S I N A

QUE PARA OBTENER EL TITULO DE

LICENCIADO EN MATEMATICAS
APLICADAS Y COMPUTACION

P R E S E N T A

HERLINDA VITE PEREZ

ASESORA: LIC. MARIA DEL CARMEN VILLAR PATIÑO

MEXICO EDO. MEX. 1998

Dedicatoria

"No es parte de
nada
porque es un suspiro
confundido
por la muchedumbre
entre los murmullos
de la
indiferencia"

A mi madre.

A la memoria de Miguel.

Agradecimientos

A Carmen Villar, por los valiosos comentarios y todo el apoyo brindado para la terminación del presente proyecto.

A Paty Carrillo y Familia, por su valiosa colaboración para la realización de éste trabajo. Por su paciencia y cariño que en todo momento me han dado.

A Miguel Ángel y Alejandro, por el apoyo incondicional que siempre me han brindado.

A mi padre, por el apoyo para la presentación de éste trabajo.

Indice

Indice

Introducción

La Ciencia de la Computación avanza día a día, implementando nuevos sistemas y elementos electrónicos que hacen aún más potente y eficaz a una computadora, más versátil a un programa y más accesible a un lenguaje.

Las computadoras han pasado por distintas etapas, desde las máquinas convencionales, que contaban con un mínimo de memoria, que sólo ofrecían exactitud y rapidez en la ejecución de un proceso hasta los sofisticados sistemas de multimedios -accesibles a casi todos los usuarios de una computadora- y robots especializados.

Los equipos computarizados, actualmente se pueden ver casi en todas partes. En las escuelas para apoyo didáctico, en los hospitales para el diagnóstico médico, en oficinas gubernamentales para agilizar y controlar trámites, en los centros de investigación explorando el espacio sideral o el fondo marino, dentro de los hogares para ayudar en la administración del gasto familiar o en juegos de video; son algunas de sus múltiples aplicaciones.

Para lograr todo esto, se han tenido que desarrollar nuevos equipos, más versátiles, capaces de adaptarse a las necesidades de las personas.

Esto ha tenido como consecuencia avances en el diseño físico, construyendo computadoras grandes, medianas y pequeñas y obviamente, en el diseño de técnicas de programación.

En el campo del diseño de software, se han desarrollado sofisticados algoritmos, principalmente dentro de la disciplina de la Inteligencia Artificial, la cual, en términos generales, se encarga de simular actos que el hombre denomina inteligentes.

Una de sus áreas, con gran aplicación práctica, son los Sistemas Expertos, que tienen la función de convertirse, como su nombre lo indica, en un experto dentro de una disciplina determinada.

Comúnmente, estos sistemas, son utilizados para efectuar tareas altamente complejas, las cuales en el pasado solo podían ser realizadas por un reducido grupo de expertos humanos.

Los Sistemas Expertos se han desarrollado en una amplia variedad de aplicaciones, tales como el diagnóstico médico, la planeación, el diseño, la instrucción, etcétera. El modo de simular el comportamiento humano por estos sistemas es utilizando conocimientos específicos de un área determinada e inferencias.

Introducción

En estos días está tomando auge una nueva técnica: la *Computación Evolutiva*.

La teoría de la selección natural o de la supervivencia del más apto, propuesta por Carlos Darwin en 1859, gobierna la adaptación evolutiva del mundo biológico.
La relación que tienen la herencia, la reproducción sexual, los cromosomas y los genes con la aptitud para sobrevivir; fue puesta de manifiesto posteriormente a través de múltiples trabajos realizados por notables investigadores de todas partes del mundo.

El reconocimiento de que el proceso evolutivo natural es un método de búsqueda y optimización robusto ha llevado al concepto de la Computación Evolutiva.

La Computación Evolutiva puede definirse como una serie de técnicas que basadas en los mecanismos de selección que utiliza la naturaleza, buscan que los algoritmos computacionales se adapten más fácilmente a los cambios que se producen en su entorno.

Entre estas técnicas se encuentran: los Algoritmos Genéticos, los Sistemas Evolutivos y la Programación Genética.

Los Algoritmos Genéticos (AG) son métodos de búsqueda basados en principios de selección natural y genética poblacional. Son los algoritmos más populares dentro del campo de la Computación Evolutiva.

Introducción

Los Sistemas Evolutivos son sistemas capaces de adquirir conocimiento de un área de aplicación determinada.

Finalmente, la Programación Genética, es una forma de lograr que las computadoras se programen para la resolución de problemas especificando "qué es lo que se tiene que hacer" y no "como se hace".

El presente trabajo, pretende ser un material de apoyo para todos los interesados en las nuevas técnicas de programación así como un aliciente para los que decidan dedicarse a la investigación en la infinidad de posibilidades del universo de la Computación.

CAPÍTULO I

ALGORITMOS GENÉTICOS

I.1 ANTECEDENTES

I.2 DEFINICIÓN

I.3 ARQUITECTURA COMPUTACIONAL

I.4 PLANTEAMIENTOS PARA EL USO DE UN ALGORITMO GENÉTICO

I.5 FUNCIONAMIENTO

I.6 VENTAJAS Y DESVENTAJAS CON RESPECTO A OTRAS TÉCNICAS DE BÚSQUEDA

I.7 AMBIENTES DE PROGRAMACIÓN

I.8 EJEMPLO DE USO

Capítulo I

I.1 Antecedentes

Entre las técnicas de búsqueda y optimización que más se están utilizando y estudiando actualmente se encuentra la de los Algoritmos Genéticos (AG) debido a los resultados que se han logrado con ellos resolviendo problemas de gran complejidad computacional en tiempos relativamente cortos.

Son los algoritmos más populares del campo de la Computación Evolutiva, un área de investigación que ha recibido una atención creciente en los últimos años.

Esta técnica, se basa en los mecanismos de selección que utiliza la naturaleza, de acuerdo a los cuales los individuos más aptos de una población son los que sobreviven al adaptarse más fácilmente a los cambios que se producen en su entorno.

Hoy, en dia, se sabe que estos cambios se efectúan en los genes de un individuo (unidad básica de codificación de cada uno de los atributos de un ser vivo), y que sus atributos más deseables (i. e., los que le permiten adaptarse mejor a su entorno) se transmiten a sus descendientes cuando éste se reproduce sexualmente.

Capítulo I

Un investigador de la Universidad de Michigan llamado John Holland estaba consciente de la importancia de la selección natural, y a fines de los '60 desarrolló una teoría que permitió incorporarla en un programa de computadora.

Su objetivo era lograr que las computadoras aprendieran por sí mismas. A la técnica que creó Holland, se le llamó originalmente "planes reproductivos", pero se hizo popular bajo el nombre de "algoritmo genético".

Los algoritmos estudiados en el campo de la Computación Evolutiva, trabajan sobre una población de individuos que representan soluciones potenciales al problema que se desea resolver.

Aplican operadores de cambio sobre la población o incorporan el uso de la selección para determinar cuáles individuos deben mantenerse en futuras generaciones y cuales se eliminan de la población.

Los individuos se adaptan al medio ambiente específico determinado por el problema, en una clara analogía con el proceso natural de evolución.

I.2 Definición

Una definición bastante completa de un Algoritmo Genético es la propuesta por John Koza[1]:

> **Es un algoritmo matemático altamente paralelo que transforma un conjunto de objetos matemáticos individuales con respecto al tiempo usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y supervivencia del más apto, y tras haberse presentado de forma natural una serie de operaciones genéticas de entre las que destaca la recombinación sexual. Cada uno de estos objetos matemáticos suele ser una cadena de caracteres (letras o números) de longitud fija que se ajusta al modelo de las cadenas de cromosomas, y se les asocia con una cierta función matemática que refleja su aptitud.**

1 Koza, J.R. "Genetic Programming -On the Programming of Computers by Means of Natural Selection-", The MIT Press, 1992.

Capítulo I

Los Algoritmos Genéticos emplean estructuras de cadenas que representan posibles soluciones para el espacio de búsqueda definido. Cada posición en la cadena contiene un carácter (usualmente con valores 0 o 1), que se comporta como un gene, ocupando una posición fija dentro de un cromosoma. El algoritmo procede partiendo de una generación inicial hacia una final en un proceso durante el cual las cadenas sufren una serie de mejoras, remplazando subcadenas para formar mejores cadenas.

Un mecanismo de selección de tipo Darwiniano que trabaja a través de mecanismos aleatorios, elige las cadenas que deben pasar a formar parte de la nueva generación. Simultáneamente, un conjunto de operadores genéticos (selección, cruce, mutación) proporcionan los mecanismos que transforman dichas cadenas para ser colocados dentro de la siguiente generación.

El operador de selección se refiere a la selección de los individuos con los que se va a trabajar. El cruce, se refiere a la copia aleatoria donde se crea un nuevo par de individuos. Y la mutación, es el operador que puede cambiar la estructura de las nuevas cadenas.

Así, aplicando un conjunto de procesos de selección, cruce y mutación, se explora la región de búsqueda, dando paso a la aparición de nuevas cadenas. Debido a que las evaluaciones son hechas a partir de la aptitud de las cadenas, la búsqueda tiende a la supervivencia de las cadenas más aptas.

I.3 Arquitectura Computacional

Desde el punto de vista de computación, un AG consta básicamente de dos partes: el sistema de datos y el sistema de control. El primero se subdivide en tres componentes:

1. Población, parte de los datos que almacena los conjuntos de cadenas.
2. Cubeta de Apareamiento, donde se reproducen las cadenas generadas.
3. Nuevos individuos, donde almacenan los nuevos individuos antes de ser integrados a la población.

El uso de estos individuos se maneja a través del sistema de control.

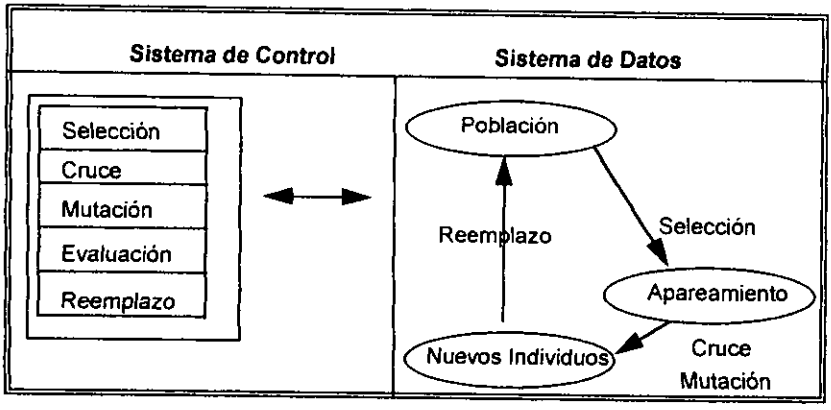


FIGURA I.1: Arquitectura computacional de un AG.

Los operadores del sistema de control se pueden clasificar en dos categorías:

- **GENERADORES**, donde se incluyen los mecanismos de Selección, Cruce y Mutación.
- **EVALUADORES**, entre los que se encuentran los Evaluadores de Aptitud y mecanismos de Reemplazo.

Los primeros se encargan de crear nuevas soluciones, mientras que los segundos tienen por tarea el evaluar y descartar soluciones.

El operador de selección es el responsable de seleccionar los individuos con los que se debe trabajar. Básicamente realiza selecciones aleatorias, equipadas con una función de probabilidad basada en la aptitud de los individuos, de modo que aquellos con mayor aptitud tengan mayor oportunidad de ser seleccionados.

Una vez seleccionados, los copia a la cubeta de apareamiento en donde las cadenas se cruzan en una posición aleatoria para crear un nuevo par de individuos. Asimismo, el operador de mutación puede cambiar la estructura de las nuevas cadenas al mutar uno o más caracteres dentro de la misma.

Cada posible solución es asociada a una aptitud dependiendo de la función objetivo definida para el problema.

Finalmente, los mecanismos de Reemplazo deciden cómo se integrarán los nuevos individuos a la población; por ejemplo, si se reemplazarán todos los individuos o solamente los menos aptos. Este ciclo es repetido hasta alcanzar el criterio de terminación definido.

I.4 Planteamientos para el uso de un algoritmo genético

Una de las aplicaciones más común de los algoritmos genéticos ha sido la solución de problemas de optimización, donde se ha mostrado su eficiencia y confiabilidad.

Sin embargo, no todos los problemas pudieran ser apropiados para esta técnica, y se recomienda en general, tomar en cuenta las siguientes características del mismo antes de intentar usarlo:

- Su espacio de búsqueda (i. e. sus posibles soluciones) debe estar delimitado dentro de un cierto rango.
- Debe existir la posibilidad de definir una función de aptitud que nos indique qué tan buena o mala es una cierta respuesta.
- Las soluciones deben codificarse de una forma que resulte relativamente fácil la implementación de las mismas.

Todos estos puntos deben tomarse en consideración, sin embargo, es conveniente poner especial atención en el primero.

Lo más recomendable es aplicar esta técnica a problemas con espacios de búsqueda discretos -aunque éstos sean muy grandes-. No obstante, también se puede intentar usar esta técnica con espacios de búsqueda continuos, pero debe existir preferentemente, un rango de soluciones relativamente pequeño.

La **función de aptitud**, no es más que la función objetivo del problema que se quiere optimizar.

El algoritmo genético únicamente maximiza, pero, cuando se requiera la minimización de un problema, ésta se puede realizar fácilmente aplicando el recíproco de la función maximizante (debe cuidarse, por supuesto, que el recíproco de la función no genere una división por cero).

Una característica que debe tener esta función es que tiene que ser capaz de "castigar" a las malas soluciones, y de "premiar" a las buenas, de manera que sean estas últimas las que se propaguen con mayor rapidez.

Una manera común de codificar las soluciones es a través de cadenas binarias, aunque se utilizan también números reales y letras.

El primero de estos esquemas, es muy popular debido a que es originalmente el propuesto por Holland[2] y además, su implementación es relativamente fácil.

[2] Holland, J. H. Creador de la técnica de los Algoritmos Genéticos (1975).

I.5 Funcionamiento

Para ilustrar el funcionamiento de un Algoritmo Genético simple, se usará el siguiente diagrama de flujo:

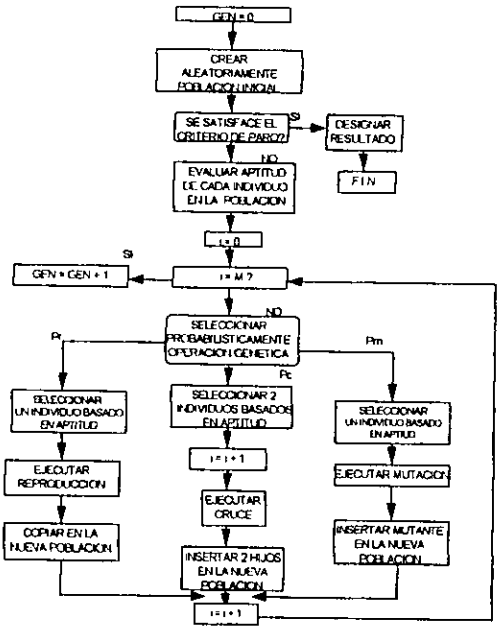


FIGURA I.2: Diagrama de flujo de un AG convencional. El índice i se refiere a un individuo en una población de tamaño M. La variable GEN es el número de generación actual.

Los parámetros primarios para el control de un AG son el tamaño de la población (M) y el número máximo de generaciones a procesar. Los parámetros secundarios, tales como: Pr, Pc y Pm son frecuencias de control de reproducción, cruce y mutación respectivamente.

Como primer paso, se genera aleatoriamente la población inicial, que deberá estar constituida por un conjunto de **cromosomas**, o cadenas de caracteres[3] que representan las soluciones posibles del problema.

A cada uno de los cromosomas de esta población se le debe aplicar la función de aptitud a fin de saber qué tan buena es la solución que se está codificando.

Una vez que se conoce la aptitud de cada cromosoma, se procede a la selección de los que se cruzarán en la siguiente generación (obviamente, se escogerá a los "mejores"). Existen dos métodos de selección que son los más comunes:

- **La Ruleta**. Este método es muy sencillo, consiste en crear una ruleta en la que cada cromosoma tiene asignada una fracción proporcional a su aptitud. Sin referirse a una función de aptitud en particular (volviendo al ejemplo), se supone que se tiene una población de 5 cromosomas cuyas aptitudes se muestran en los valores de la siguiente tabla:

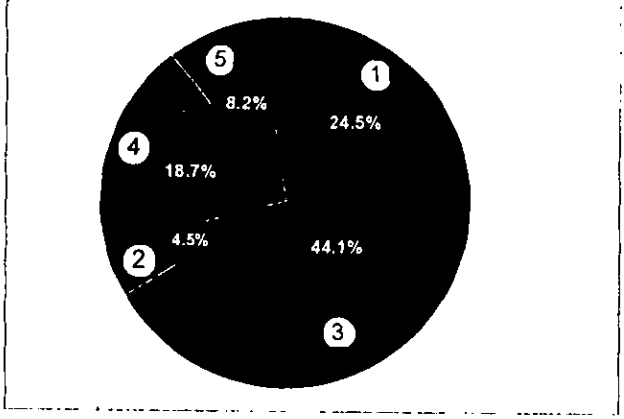
[3] *En la práctica suelen usarse cadenas binarias para representar a los cromosomas, por lo que éstas suelen* implementarse como cadenas de bits (usando el tipo booleano de Pascal o el modificador unsigned de C), pero también puede recurrirse (y se hace) al uso de letras y/o números naturales.

CROMOSOMA No.	CADENA	APTITUD	% DEL TOTAL
1	11010110	254	24.5
2	10100111	47	4.5
3	00110110	457	44.1
4	01110010	194	18.7
5	11110010	85	8.2
TOTAL	-	1037	100.0

FIGURA I.3 VALORES DE EJEMPLO PARA ILUSTRAR LA SELECCIÓN MEDIANTE RULETA

Con los porcentajes mostrados en la cuarta columna de la tabla se puede elaborar la ruleta de la figura I.4. Esta ruleta se gira 5 veces para determinar a los individuos que serán seleccionados. Como a los individuos que se consideran más aptos, se les asigna un área mayor en la ruleta, se espera que sean seleccionados un número mayor de veces que los menos aptos.

FIGURA I.4 RULETA QUE REPRESENTA LOS VALORES DE APTITUD DE LA TABLA.



El Torneo. La idea de este método es muy simple. Se baraja la población y después se hace competir a los cromosomas que la integran en grupos de tamaño predefinido (normalmente compiten en parejas) en un torneo en el que resultarán ganadores aquéllos que tengan valores de aptitud más altos. Si se efectúa un torneo binario (i. e., competencia por parejas), entonces la población debe ser barajeada dos veces. Se debe notar, que esta técnica garantiza la obtención de múltiples copias del mejor individuo entre los progenitores de la siguiente generación (si se efectúa un torneo binario, el mejor individuo será seleccionado dos veces).

Cuando la selección se ha realizado, se procede entonces a la reproducción sexual o cruza de los individuos que han sido seleccionados. En cada etapa, los sobrevivientes intercambian material cromosómico y sus descendientes formarán la población de la siguiente generación.

Las dos maneras de reproducción sexual son: uso de un punto único de cruza y uso de dos puntos de cruza.

Cuando se usa un solo punto de cruza, éste se escoge de forma aleatoria sobre la longitud de la cadena que representa el cromosoma, a partir de él se realiza el intercambio de material de los dos individuos, tal como se muestra en la siguiente figura.

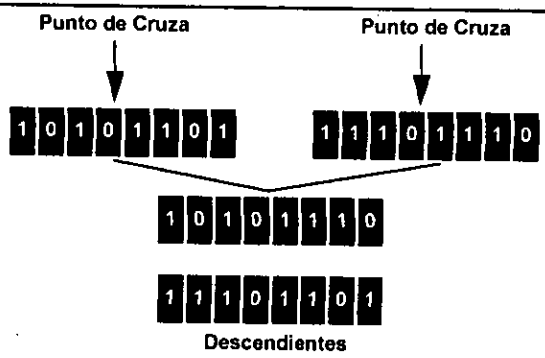


FIGURA I.5: Uso de un solo punto de cruza entre dos individuos. Note que cada pareja de cromosomas da origen a dos descendientes para la siguiente generación. El punto de cruza puede ser cualquiera de los dos extremos de la cadena, en cuyo caso no se realiza la cruza.

En el caso de usar dos puntos de cruza, se procede de manera similar, pero en este caso el intercambio se realiza de la manera como se muestra en la siguiente figura.

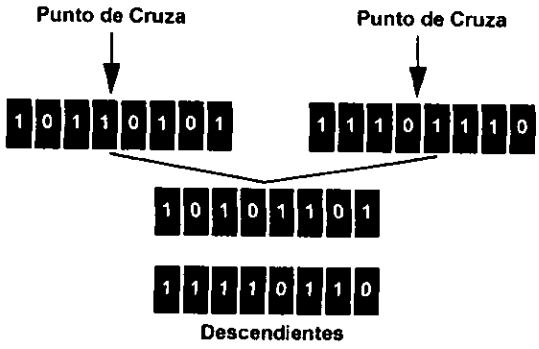


FIGURA I.6: Uso de dos puntos de cruza entre dos individuos. Observe cómo en este caso se mantienen los genes de los extremos y se intercambian los del centro. También aquí existe la posibilidad de que uno o ambos puntos de cruza se encuentren en los extremos de la cadena, en cuyo caso se hará una cruza usando un solo punto, o ninguna cruza, según corresponda.

Normalmente la cruza se maneja dentro de la implementación del algoritmo genético como un porcentaje que indica con qué frecuencia se efectuará. Esto significa, que no todas las parejas de cromosomas se cruzarán, sino que habrá algunas que pasarán intactas a la siguiente generación.

De hecho existe una técnica, desarrollada hace algunos años, en la que el individuo más apto a lo largo de las distintas generaciones no se cruza con nadie, y se mantiene intacto hasta que surge otro individuo mejor que él, que finalmente lo desplazará.

Dicha técnica, es llamada elitismo.

Existe otro operador, además de la selección y la cruza, llamado mutación. El cual realiza un cambio a uno de los genes de un cromosoma elegido aleatoriamente. Cuando se usa una representación binaria, un bit se sustituye por su complemento (un cero cambia en uno y viceversa). Este operador permite la introducción de nuevo material cromosómico en la población, tal y como sucede con sus equivalentes biológicos.

Al igual que la cruza, la mutación se maneja como un porcentaje que indica con qué frecuencia se realizará, aunque se distingue de la primera por ocurrir mucho más esporádicamente (el porcentaje de cruza, normalmente es de más del 60%, mientras que el de mutación comúnmente nunca supera el 5%).

Si se conociera de antemano la respuesta a la que se debe llegar, entonces detener el algoritmo genético sería algo trivial. Sin embargo, esto casi nunca es posible, por lo que normalmente se utilizan dos criterios principales de detención:

- Correr el algoritmo genético durante un número máximo de generaciones.
- Detenerlo cuando la población se haya estabilizado.(i. e., cuando todos o la mayoría de los individuos tengan la misma aptitud).

Los tres pasos para ejecutar un AG se pueden resumir como:

1. Crear aleatoriamente una población inicial de individuos de cadenas.
2. Ejecutar iterativamente los siguientes pasos en la población de cadenas hasta que se satisfaga el criterio de paro:
   - Evaluar la aptitud de cada individuo en la población
   - Crear una nueva población aplicando las dos primeras operaciones genéticas:

     i) Copiar una cadena existente a la nueva población

     ii) Crear una cadena nueva por recombinación genética, escogiendo aleatoriamente dos partes de las cadenas existentes.

     iii) Crear una nueva cadena de una cadena existente mutando aleatoriamente un caracter en una posición en la cadena.
3. El mejor individuo (cadena) que aparezca en alguna generación se desgna como el resultado del AG para la corrida. Este resultado representa una solución o una solución aproximada al problema.

I.6 Ventajas y desventajas con respecto a otras técnicas de búsqueda

- **VENTAJAS:**
- No necesitan conocimientos específicos sobre el problema que intentan resolver.
- Operan de forma simultánea con varias soluciones, en vez de trabajar en forma secuencial, como las técnicas tradicionales.
- Cuando se usan para problemas de optimización -maximizar una función objetivo- resultan menos afectados por los máximos locales (falsas soluciones) que las técnicas tradicionales.
- Resulta sumamente fácil ejecutarlos en las modernas arquitecturas, masivamente paralelas.
- Usan operadores probabilísticos, en vez de los típicos operadores determinísticos, de las otras técnicas.
- **DESVENTAJAS:**
- Pueden tardar mucho en converger, o no converger en absoluto, dependiendo en cierta medida de los parámetros que se utilicen (tamaño de la población, número de generaciones, etcétera).
- Pueden converger prematuramente debido a una serie de problemas de diversa índole.

I.7 Ambientes de programación

Hoy en dia, existe un gran número de ambientes de programación disponibles en el mercado para experimentar con los algoritmos genéticos.

Pueden distinguirse tres clases de ambientes de programación:

1. SISTEMAS ORIENTADOS A LAS APLICACIONES. Son esencialmente "cajas negras" para el usuario, puesto que ocultan todos los detalles de implementación. Sus usuarios - normalmente neófitos en el área- los utilizan para un cierto rango de aplicaciones diversas, pero no se interesan en conocer la forma en que éstos operan. Ejemplos de este tipo de sistemas son: Evolver (Axcelis, Inc.) y XpertRule GenAsys (Attar Software).
2. SISTEMAS ORIENTADOS A LAS ALGORITMOS. Soportan algoritmos genéticos específicos, y se subdividen en:
    - SISTEMAS DE USO ESPECIFICO. Contienen un solo algoritmo genético, y se dirigen a una aplicación en particular. Algunos ejemplos son: Escapade (Frank Hoffmeister), GAGA (Jon Crowcroft) y Génesis (John Grefenstette).
    - BIBLIOTECAS. Agrupan varios tipos de algoritmos genéticos, y diversos operadores (e. i., distintas formas de realizar la cruza y la selección). Algunos ejemplos representativos de este grupo son: Evolution Machine (H.M. Voigt y J. Born) y OOGA (Lawrence Davis).

En estos sistemas se proporciona el código fuente para que el usuario -comúnmente un programador- pueda incluir el algoritmo genético en sus propias aplicaciones.

3. CAJAS DE HERRAMIENTAS. Proporcionan muchas herramientas de programación, algoritmos y operadores genéticos que pueden aplicarse en una enorme gama de problemas. Normalmente se subdividen en:
    - SISTEMAS EDUCATIVOS. Ayudan a los usuarios principiantes a introducirse de forma amigable a los conceptos de los algoritmos genéticos. GA Workbench (Mark Hughes) es un buen ejemplo de este tipo de ambiente.
    - SISTEMAS DE PROPÓSITO GENERAL. Proporcionan un conjunto de herramientas para programar cualquier algoritmo genético y desarrollar cualquier aplicación. Tal vez el sistema más conocido de éste tipo es: Splicer (NASA).

I.8 Ejemplo de uso

OPTIMIZACIÓN DE UN DISEÑO DE INGENIERÍA USANDO ALGORITMOS GENÉTICOS.

Este ejemplo, ilustra la representación gráfica que se usa frecuentemente en aplicaciones prácticas de los Algoritmos Genéticos a problemas de Optimización.

El problema, es un problema de Ingeniería descrito por Goldberg y Samtani en *Induction: Process of Inference, Learning, and Discovery*, MIT, 1986.

La figura I.7 muestra una estructura de 10 componentes las cuales son 10 secciones tranversales etiquetadas con A_1, A_2,..., A_{10}.

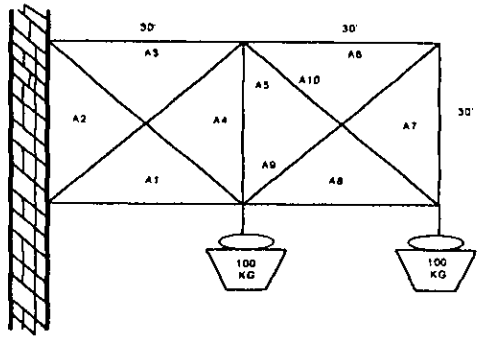


FIGURA I.7: ESTRUCTURA DE 10 COMPONENTES.

La estructura esta soportada por una pared en el lado izquierdo y debe soportar dos cargas como se muestra en la figura I.7. Además, la tensión de cada componente debe estar en un rango permitido representada por una tensión muy estrecha para cada componente.

El objetivo es encontrar el área de corte transversal (sección) para cada componente de la estructura así como mimimizar el peso total (costo) de el material usado en la construcción de la misma.

Este problema requiere una búsqueda en un espacio de 10 dimensiones de números reales para la combinación de valores A_1, A_2,..., A_{10} que tenga la mejor aptitud (i.e., menor costo o peso).

El primer paso para usar un AG, es seleccionar la representación de un esquema.

Un esquema muy popular es la representación de un conjunto de números reales como cadenas binarias de longitud fija en el cual cada número real es asociado con parte de la cadena completa. Goldberg y Samtani decidieron representar las 10 secciones de corte mediante una cadena de 40 bits. También, creyeron conveniente, representar un corte transversal con un valor de 0.1 pulgada cuadrada que debe representarse por 4 bits: 0000 y que una sección fuese igual a 10.0 pulgadas cuadradas que se representan por 4 bits: 1111.

Cada uno de los 14 bits restantes se debe codificar con valores intermedios apropiados para el área de corte. Cada uno de estos 14 bits restantes se codifican con valores apropiados intermedios para el área de corte transversal (sección).

La figura I.8 un cromosoma de longitud 40 que representa a la estructura. Los primeros 4 bits de la cadena son para A_1. Éstos 4 bits permiten a la primer componente de la estructura tomar uno de 16 diferentes valores posibles de una sección.

Por ejemplo, la primer área: A_1, se codifica como 0010 y equivale a 0.66 pulgadas cuadras. Cada una de las 9 áreas restantes se representan de modo similar usando 4 bits.

0010	1110	0001	0011	1011	0011	1111	0011	0011	1010

FIGURA I.8: CROMOSOMA DE LONGITUD 40 QUE REPRESENTA A LA ESTRUCTURA DE 10 COMPONENTES.

En la selección del esquema para la representación de éste problema, Goldberg y Samtani, usaron su conocimiento del problema en particular para seleccionar la granularidad de los diferentes valores posibles para las áreas de corte.

En resumen, el esquema usado por Goldberg y Samtani, involucra un alfabeto de tamaño 2 (i.e., k=2), cromosomas de longitud 40 (i.e., L=40), y el mapeo entre los 10 valores reales de las áreas de corte y los cromosomas de 40 bits.

La selección de K, L y el mapeo, constituye el primer paso para el uso de un AG con cadenas de carácteres de longitud fija. El espacio de búsqueda es de tamaño 2^{40}, lo que es aproximadamente 10^{12}.

El segundo paso, consiste en identificar la medida de aptitud que inquiera que tan bueno es un elemento de la estructura que se encuentre representado en la cadena de 40 bits al ejecutar el algoritmo. Para Goldberg y Samtani, la medida de aptitud de un punto dado en el espacio de búsqueda (i.e., un diseño dado para la estructura) debería ser el costo total del material para la estructura.

Si un punto en el espacio, viola uno o más de las 10 condiciones de tensión, la aptitud, seria el costo total de el material más un valor de pena o castigo, por la imposibilidad que representa el punto. Para este problema, la aptitud es una función no lineal de 10 variables.

El tercer paso, consiste en la selección de los parámetros y variables de control para el algoritmo.

Los dos parámetros más importantes son el tamaño de la población (M) y el máximo número de generaciones a ser procesadas (G). En la solución de éste problema, Goldberg y Samtani usaron una población de M=200 cadenas de bits de longitud L=40 y un máximo de generaciones de G=40.

El cuarto paso, es decidir, la elección de el método de terminación de una ejecución y el método de designación de resultados.

Goldberg y Samtani, terminarón la ejecución del algoritmo, después de generar el número máximo de generaciones (G=40) y designaron como mejor resultado el que obtuvieron en la última corrida del algoritmo.

Una vez que éstos cuatro pasos se realizan, el AG procede en un domio independiente y trata de resolver el problema. El objetivo del AG es buscar su multidimensionalidad, el espacio de búsqueda no lineal para el punto con aptitu globalmente óptima (i.e., el peso o el costo).

En la práctica, se usó una población de tamaño M=200. Se ejecutaron varias corridas, en las cuales alrededor de 8000 individuos fueron procesados en cada corrida (i.e., 40 generaciones de 200 individuos). En cada corrida, se obtuvo un diseño factible para las 10 componentes de la estructura por lo cual el costo total del material se consideró dentro del 1% establecido para la mejor solución conocida.

El número de individuos que debe procesarse al resolver un problema dado, muchas veces usa como medida de proceso (computación) la más alta especificación asociada con el AG.

CAPÍTULO II

FUNDAMENTOS SOBRE SISTEMAS EVOLUTIVOS

II.1 ANTECEDENTES

II.2 DEFINICIÓN

II.3 ARQUITECTURA

II.4 ÁREAS DE APLICACIÓN

II.5 MÉTODO DE DESARROLLO

II.5.1 DETECCIÓN DEL ÁREA DE APLICACIÓN

II.5.2 DETECCIÓN DE EJEMPLOS SIGNIFICATIVOS

II.5.3 ESTUDIO DEL LENGUAJE

II.5.4 DETECCIÓN DE UNIDADES LÉXICAS

II.5.5 OBTENCIÓN DE GRAMÁTICA CANÓNICA

II.5.6 OBTENCIÓN DE GRAMÁTICA GENERALIZADA

II.5.7 APLICACIÓN DE REGLAS TRANSFORMACIONALES

II.5.8 CONSTRUCCIÓN DE LA BASE DE CONOCIMIENTOS

II.5.9 CONSTRUCCION DEL MECANISMO DE DIALOGO USUARIO SISTEMA

Capítulo II

II.1 Antecedentes

Los Sistemas Expertos, que se desarrollan en una amplia variedad de aplicaciones, como el diagnóstico médico, la planeación, el diseño, la instrucción, etcétera, el modo de simular el comportamiento humano por estos sistemas utiliza conocimientos específicos de un área determinada e inferencias.

Dentro del área de la Computación Evolutiva se desarrollan los Sistemas Evolutivos, aplicación que va dirigida a la automatización de métodos de programación de tipo inductivo, mediante los cuales sea relativamente fácil identificar el conjunto de patrones o reglas generales de un problema a partir de ejemplos particulares.

Uno de los aspectos fundamentales, que caracterizan a estos sistemas, es el que son capaces de aprender palabras aisladas, reglas y significados sin que hayan sido programados de antemano, lo cual permite que el sistema actualice constantemente su Base de Conocimientos.

Otra característica importante, es que el desarrollo de estos sistemas se basa en la Teoría de los Lenguajes Formales y en métodos de Reconocimiento Sintáctico de Patrones.

Capítulo II

Los Sistemas Evolutivos están orientados a facilitar la comunicación entre usuario y computadora en un lenguaje lo más natural posible, así como en al manejo por parte de la máquina de formas o patrones más que de datos o conocimientos específicos.

La manipulación de requerimientos se lleva a cabo en un lenguaje natural restringido, similar al que utiliza el usuario en su área de aplicación.

El nombre, así como la idea de estos sistemas, se ha conformado con el paso del tiempo, pasando por Sistemas que se Adaptan hasta llegar al nombre y concepto con el que se les conoce ahora: Sistemas Evolutivos.

II.2 Definición

Un Sistema Evolutivo, es un sistema capaz de adquirir conocimientos de un área especifica y almacenarlos dentro de una Base de Conocimientos (BC) del sistema. La información que adquiere un sistema en la etapa de aprendizaje es utilizada para solucionar problemas que tienen una estructura sintáctica igual a los problemas aprendidos.

Se habla de estructuras sintácticas, porque -como se verá más adelante- estos sistemas se apoyan en gramáticas para representar y manipular las estructuras de los problemas.

Para definir estos sistemas es importante indicar que tienen dos funciones principales, las cuales cubren, por un lado, la función de adquirir y almacenar el conocimiento y por otro la de administrarlo o explotarlo.

La función de aprendizaje se encarga de todo lo relacionado con los problemas que no son conocidos por el sistema, para lo cual se le debe proporcionar información sobre vocabulario, estructura sintáctica y semántica de los problemas que no son conocidos. Toda esta información es interpretada por el sistema y almacenada en su Base de Conocimientos.

La información se proporciona a través de ejemplos cuya estructura es similar al problema no conocido, lo cual le permite generalizar y ampliar su conocimiento.

También es importante señalar que estos sistemas aprenden a resolver problemas de una manera gradual, lo que significa, que los conocimientos se dan en niveles de dificultad, pasando de niveles sencillos hasta más complejos.

Cuando el sistema ha aprendido a resolver problemas, tiene la capacidad de atender otros cuya estructura sea la misma que los problemas almacenados.

En esta etapa, el sistema analiza todos los problemas que son ingresados, generando como primer paso una traducción a un formato que el sistema pueda manejar, el cual es similar al que tiene almacenado en su base de conocimientos.

Posteriormente, compara los problemas ingresados contra las estructuras que tiene almacenadas, dando solución a los problemas que le son conocidos y pasando el control a la función de aprendizaje en aquellos problemas que no conoce.

Las dos funciones principales de un Sistema Evolutivo cuentan con un nombre que las identifica plenamente con las tareas que realizan; siendo el nombre de la función de aprendizaje: **CONSTRUCTOR** del sistema y el de la etapa de explotación del conocimiento: **ADMINISTRADOR** del sistema.

II.3 Arquitectura y funcionamiento

De acuerdo a la definición de Sistemas Evolutivos, un sistema de esta clase se puede dividir en dos ciclos, los cuales determinan su arquitectura. El CONSTRUCTOR y el ADMINISTRADOR pueden verse como dos subsistemas, ya que cada uno realiza funciones específicas.

Por ejemplo, en la función que realiza el ADMINISTRADOR cuando un problema es conocido o el caso de los algoritmos de inferencia gramatical en el CONSTRUCTOR, sólo por mencionar dos de las funciones que forman parte de cada uno de ellos.

El CONSTRUCTOR utiliza todas las herramientas necesarias que permiten al sistema adquirir conocimiento y generalizarlo. En el ADMINISTRADOR se determina la manera de utilizar el conocimiento adquirido, así como la función que permite al sistema saber cuando debe pasar a una etapa de aprendizaje.

En la figura siguiente se describe la arquitectura general de los Sistemas Evolutivos.

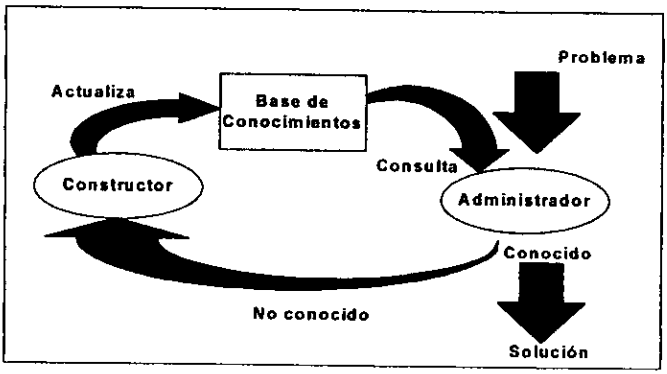


FIGURA II.1: Arquitectura de un Sistema Evolutivo.

Cuando ingresa un problema al sistema, se procesa primeramente en el ADMINISTRADOR, el cual se encarga de hacer las consultas que sean necesarias a la Base de Conocimientos y así determinar si el problema es conocido o no.

En el momento que se determina que el problema es conocido, el ADMINISTRADOR se encarga de generar la solución al problema, haciendo uso de reglas transformacionales, rutinas semánticas u otras técnicas similares. Pero cuando el problema no es conocido cede la tarea al CONSTRUCTOR, éste se encarga de adquirir todos los conocimientos necesarios acerca del problema y poder así, actualizar la Base de Conocimientos con la nueva Información.

Para efectuar esto, el CONSTRUCTOR pide varios ejemplos, los cuales deben tener las mismas características que el problema original, para que de aqui, el CONSTRUCTOR pueda generalizar y por ende manejar su conocimiento.

Por último, cuando el CONSTRUCTOR termina de analizar el nuevo problema, así como de actualizar su base de conocimientos, cede el control nuevamente al ADMINISTRADOR, el cual, ahora está listo para resolver el problema que le era desconocido anteriormente.

Las dos grandes partes de un Sistema Evolutivo están en permanente comunicación, lo cual hace que el sistema se actualice constantemente, convirtiéndose así, en la función principal que define a estos sistemas.

II.4 Áreas de aplicación

Los Sistemas Evolutivos pueden ser aplicados en cualquier campo donde se puedan extraer estructuras a partir de ejemplos. Para representar estas estructuras se pueden utilizar gramáticas, las cuales servirán al sistema en el momento que se quiera resolver algún problema, el cual deberá estar contenido dentro de las gramáticas.

Así, se tiene que los Sistemas Evolutivos pueden ser desarrollados en áreas tales como:

- Matemáticas
- Generadores de sistemas
- Visión
- Sistemas de purificación de agua

II.5 Método de desarrollo

La necesidad de desarrollar más y mejores cosas en menos tiempo, ha hecho que el hombre ordene sus pasos y técnicas. Este orden que se sigue para desarrollar algo, se denomina método o procedimiento, cuya finalidad es servir de guía para efectuar alguna actividad de modo eficaz.

En el ámbito de desarrollo de sistemas existen varios métodos para realizar un sistema de una manera ordenada y sencilla, los cuales en su gran mayoría se resumen en actividades similares. El primer paso es elaborar un análisis de los aspectos que están relacionados con el sistema, otro es elaborar un diseño que contemple la forma en que se construirá el sistema y el último, es la programación y posterior implementación.

Para el desarrollo de Sistemas Evolutivos, se tiene una serie de etapas de análisis, diseño y programación, dentro de un método conocido con el nombre de Programación Dirigida por Sintaxis.

La diferencia que existe en el desarrollo de Sistemas Evolutivos, son las herramientas que se utilizan, ya que proponen que un sistema de información puede ser representado por gramáticas.

Capítulo II

Desarrollando el método de programación dirigida por sintaxis se obtiene un sistema evolutivo, este método consta de las siguientes etapas:

- Detección del área de aplicación.
- Detección de ejemplos significativos.
- Estudio del lenguaje.
- Detección de unidades léxicas.
- Obtención de una gramática canónica.
- Obtención de una gramática generalizada.
- Aplicación de reglas transformacionales.
- Construcción de la base de conocimientos.
- Construcción del mecanismo de diálogo usuario-sistema-usuario.

A continuación, se da una breve descripción de cada una de ellas.

II.5.1 Detección del área de aplicación

El primer paso para desarrollar un sistema, es definir y delimitar el área dentro de la cual el sistema va a operar.

En este primer paso, es importante considerar las características generales del área de aplicación, así como tener una breve descripción de cómo funciona ésta.

Es importante definir el alcance del sistema, lo cual, quiere decir, que si en el área de aplicación se manejan varios mecanismos para comunicarse, tales como: lenguaje oral, imágenes u otros, se debe definir si el sistema tendrá la capacidad de manejar todos los mecanismos o sólo uno o algunos de ellos.

Ya que esto, en un momento dado, puede estar condicionado al equipo donde se desarrollará el sistema o simplemente a restricciones de la tecnología actual.

II.5.2 Detección de ejemplos significativos

Dado que se tiene alguna área de aplicación o algún problema a resolver, el primer punto importante consiste en la detección del lenguaje o lenguajes involucrados, lo cual se logra mediante el análisis de ejemplos.

El objeto de analizar ejemplos del área de aplicación es para encontrar los diferentes lenguajes que se utilizan.

En esta etapa se recomienda hacer una combinación de los diferentes lenguajes, ya que en una situación especifica, es mejor representar una figura combinándola con un texto alusivo a ella, que representar la figura y el texto de una manera aislada.

En la siguiente tabla, podemos observar ejemplos significativos en algunas áreas:

ÁREA DE APLICACIÓN	EJEMPLOS DE LENGUAJE
Procesamiento de imágenes	
Matemáticas	15x Dx 5y Dy
Generador de Sistemas	Lista los empleados que tienen un sueldo mayor a $5000.00. Calcula el factor de ocupación en el tramo ACA-MEX.
Bases de datos de reservaciones	Dame la cantidad de asientos disponibles en la ruta MEX-QRO del día 20 de Mayo de 1997. Reserva un asiento para Laura Rangel en el vuelo 940 del día 20 de Mayo de 1997

FIGURA II.2: Algunas de las áreas de aplicación para un Sistema Evolutivo.

II.5.3 Estudio del lenguaje

Como se observa, de los ejemplos anteriores, se pueden obtener los diferentes tipos de lenguaje que se utilizan.

Es importante considerar que el concepto de lenguaje es muy amplio, ya que cualquier mecanismo en el que se encuentre un conjunto de elementos (alfabeto), y sobre los cuales se puede aplicar un conjunto de reglas para relacionarlos (sintaxis) y asociarle un significado (semántica), se puede decir que cuenta con un lenguaje. En su momento pueden existir lenguajes de múltiples tipos, tales como:

- Simbólicos
- Visuales
- Trayectorias
- Orales
- Escritos

En el desarrollo de estos sistemas, es importante, no restringirse a lenguajes tradicionales, tales como: lenguajes de programación, lenguajes manejadores de bases de datos u otros similares, ya que en el universo de la comunicación existen múltiples mecanismos, los cuales pueden representar algún problema de una manera más adecuada.

Como posibles ejemplos de oraciones de los diversos tipos de lenguaje se tienen:

A) En el diagnóstico médico, la conversación entre paciente y médico.

B) En visión, la imagen a reconocer.

C) En juegos de tablero, la trayectoria de la pieza. En este caso la idea es que la trayectoria de una pieza en el tablero se puede ver como una imagen visual y por tanto aplicarle las herramientas del ejemplo anterior.

II.5.4 Detección de unidades léxicas

En esta etapa se estudia el mecanismo de representación del lenguaje. Para lo cual, primero se identifican las unidades con significado mínimo o unidades léxicas, por otro lado, se clasifican de acuerdo a un tipo, el cual esta determinado por el tipo de sistema de que se trate.

En el área de procesamiento de imágenes podemos tener la siguiente figura:

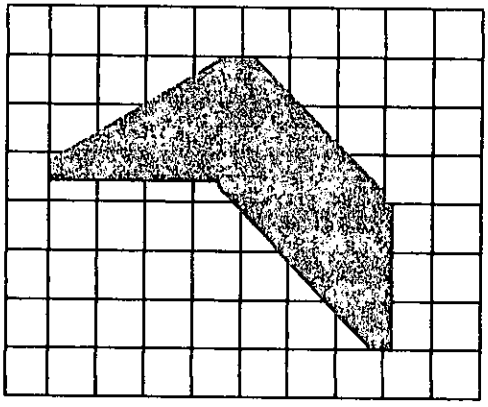


FIGURA II.3: Imágen a procesar.

En esta imagen, se identifican las unidades con significado mínimo siguientes:

/ DIAGONAL DERECHA

\ DIAGONAL IZQUIERDA

| LINEA VERTICAL

- LINEA HORIZONTAL

El mecanismo que se utiliza para interpretar figuras con este tipo de unidades léxicas, se observa en la siguiente figura:

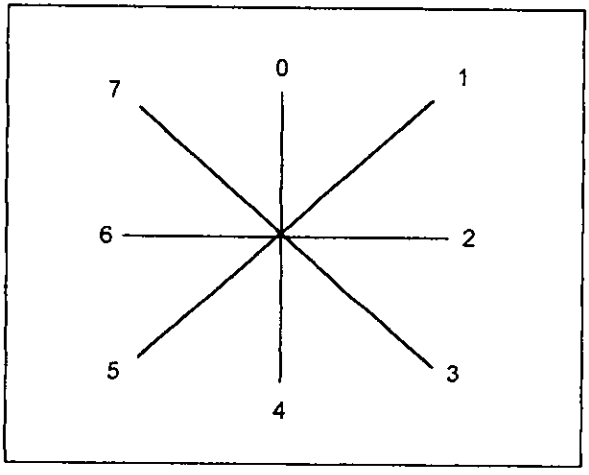


FIGURA II.4: Mecanismo para interpretar una imágen a procesar.

En el lenguaje español se puede observar el siguiente ejemplo:

DIME LA CANTIDAD DE BOLETOS VENDIDOS DEL VUELO 904, DEL DÍA 8 DE MARZO DE 1997.

De la oración anterior, se puede generar la siguiente representación:

a b c d e c d c f d c f

Las unidades léxicas o unidades mínimas con significado, dependerán directamente del tipo de lenguaje que se este representando, y en un momento dado, pueden variar de un lenguaje a otro, ya que mientras en un lenguaje se consideran ciertas unidades léxicas, en otro lenguaje pueden no tener significado esas mismas unidades.

II.5.5 Obtención de Gramática Canónica

Ya que se tiene el mecanismo de representación del lenguaje, se procede a analizar varios ejemplos, sustituyéndose cada unidad léxica por su código correspondiente, obteniendo por cada ejemplo una oración canónica. Al agrupar las oraciones canónicas, se obtiene una gramática, la cual nos representa la estructura sintáctica de los ejemplos. A esta gramática se le conoce como Gramática Canónica.

Las oraciones canónicas representan los patrones generales o estructura de una oración tradicional.

Por ejemplo, del siguiente grupo de oraciones comunes:

- El gato es negro
- El oso es gris
- La gallina es blanca

pueden ser representadas por la siguiente oración canónica:

a b c d

donde:

a- representa un artículo: El, La

b- representa el sujeto: gato, oso, gallina

c- representa el verbo: es

d- representa el adjetivo: negro, gris, blanca

La oración canónica representa la estructura de una oración, no a las unidades léxicas en sí. La manera de formar una oración canónica es agrupando unidades léxicas de acuerdo a un tipo de unidad.

La Gramática Canónica es el conjunto de oraciones canónicas. Y se le da el nombre de Gramática porque representa la estructura de un tipo de oraciones del lenguaje.

II.5.6 Obtención de Gramática Generalizada

Una vez que se han detectado los componentes del lenguaje (unidades léxicas) y se ha llegado a la Gramática Canónica, el siguiente paso es mejorar esa gramática aplicando algoritmos de Inferencia Gramatical.

Esta etapa es muy importante para el método, ya que permite encontrar los patrones o formas generales de un lenguaje a partir de ejemplos particulares. Con lo que se obtiene un método de programación de tipo inductivo.

II.5.7 Aplicación de reglas transformacionales

Hasta ahora, se ha visto que con una gramática se puede representar la estructura sintáctica de un problema, pero además de eso, también es necesario tener la solución de un problema (manejo de semántica). Para lo cual se puede hacer uso de reglas transformacionales, las cuales consisten en representar una estructura sintáctica por otra, sin perder información del problema original.

Para realizar la transformación se hacen equivalencias entre Gramáticas Generalizadas, a las cuales se hace referencia en la etapa del Sistema Evolutivo que da solución a un problema.

Además de las reglas transformacionales existen otras técnicas para el manejo de la semántica. Por ejemplo, en el caso de los Compiladores, se hace introduciendo atributos en las producciones de la gramática. Estos atributos son llamados a rutinas semánticas (donde una rutina semántica es una rutina que lleva a cabo alguna acción).

II.5.8 Construcción de la Base de Conocimientos

Después de obtener algunas de las diferentes representaciones de un problema: la representación léxica, sintáctica y semántica, se procede a almacenar esta información dando lugar a la Base de Conocimientos del sistema.

La estructura de la Base de Conocimientos puede variar de un sistema a otro, ya que la información que sea necesario almacenar acerca de las unidades léxicas o vocabulario, así como la manera como se manejan las gramáticas y la semántica dentro del sistema determinan directamente la estructura que tendrá la Base de Conocimientos.

II.5.9 Construcción del Mecanismo de diálogo usuarios-sistema

Cuando se han construido todas las partes que forman el sistema, es necesario construir un mecanismo que permita comunicar al usuario con todo el sistema.
Este mecanismo recibe el nombre de diálogo Usuario-Sistema y debe contemplar los diferentes tipos de lenguaje que maneja el sistema de información, considerando las características del equipo en el que se desarrollará.

El mecanismo de diálogo se desarrolla en un lenguaje natural restringido, ya que uno de los objetivos de este mecanismo es facilitar la comunicación entre el sistema y sus usuarios.

Con el uso de lenguaje natural se trata de evitar los tecnicismos que complican la comunicación.

Al mecanismo de diálogo no se le debe ver como una etapa aislada del sistema evolutivo, sino por el contrario, se le debe ver como un dispositivo que se encuentra inmerso en todo el sistema.

Como ejemplo de esta técnica, se cita el trabajo realizado por: Caballero Cortés Cesar. "Sistema Evolutivo De Algebra". Tesis de la Licenciatura en Ciencias de la Informática. Departamento de Computación, UPIICSA, I.P.N., 1990.

Una aplicación de estos sistemas es THE GUARDIAN, un programa de cuidados médicos intensivos para enfermos graves. Este software recibe información del paciente directamente de la unidad de cuidados intensivos e indirectamente de su expediente y de sus resultados clínicos (rayos X, laboratorio, etc.). Con estos datos, Guardian elabora diagnósticos y recomienda procedimientos diversos con base en algoritmos elaborados con funciones de los doctores humanos, reconocedores de patrones y sistemas capaces de sugerir un procedimiento aún cuando no exista un diagnóstico exhaustivo del problema.

También existen diversos grupos de investigadores que se enfocan al estudio y desarrollo de estos sistemas como: The Oz Project at Carnegie Mellon University y The Autonomous Agent Group, en el MIT, que se abocan más al estudio de agentes que pueden aprender, colaborar y planificar; y The Cognition and Affect Project de la Universidad de Birmingham que investiga agentes inteligentes que tienen propósitos y motivaciones.

Finalmente, para consultar aplicaciones de este campo, se puede accesar: Adaptive Intelligent Systems: http://ksl-web.stanford.edu/projects/BB1/.
Éste, es un proyecto interdisciplinario que busca respuestas a como lidiar con la inestabilidad, la impredicibilidad y el dinamismo que abunda en el mundo y la conducta humana, que no siempre es compatible con las bien organizadas computadoras y programas.

Así como el proyecto Aibots, que son aplicaciones para robots móviles: http://ksl-web.stanford.edu/projects/aibots/Nomad-200.html.

CAPÍTULO III

PROGRAMACIÓN GENÉTICA

III.1 Antecedentes

Una de las metas de los investigadores en ciencias computacionales ha sido utilizar las computadoras en tareas que realmente ayuden al ser humano de una manera más participativa e inteligente.

En la solución a este tipo de dilemas, el paradigma de Programación Genética (PG) surge como una metodología con la que se formaliza el procedimiento de búsqueda de soluciones desde el punto de vista de la inducción de programas, que busca encontrar un programa de computadora dentro de un espacio posible de programas de computadora.

A lo largo de este capitulo se darán los conceptos básicos e implementación de ésta técnica.

III.2 Definición

El paradigma de Programación Genética propuesto por John Koza[4], es una extensión de los algoritmos genéticos que difiere de éstos en la forma en que representa a los individuos de la población, pues utiliza programas de computadora en lugar de cadenas de longitud fija.

La meta de la PG es lograr que las computadoras aprendan a resolver problemas sin ser explícitamente programadas, la PG trata de generar soluciones a problemas a partir de la inducción de programas. El programador no especifica el tamaño, forma y complejidad estructural de los programas-solución sino que los programas evolucionan hasta generar soluciones satisfactorias.

Tradicionalmente, la investigación en los campos de inteligencia artificial, sistemas de auto-mejora y auto-organización, aprendizaje de máquina e inducción en general, utiliza métodos correctos, consistentes, justificables, ciertos (deterministas), metódicos, parsimoniosos y decisivos (que tengan un punto de terminación bien definido). Una razón del escepticismo inicial de las personas que abordan la PG es que es difícil imaginar que estos siete conceptos, que están virtualmente integrados en el entrenamiento y pensamiento humano, no deben ser usados para resolver todos los problemas, ya que han jugado roles invaluables en la solución exitosa en muchos problemas de ciencias, ingeniería y matemáticas.

[4] *Ibid, página 145.*

Dado que las ciencias de la computación están fundadas en la lógica, es especialmente difícil para sus practicantes imaginar que estos siete principios guía no deban usarse para resolver cada problema.

Sin embargo, en la PG no se utilizan en el sentido tradicional sino que se emplean los principios de la naturaleza.

III.3 Inducción de programas

Dentro del espacio de posibles programas de computadora, la inducción de programas involucra el descubrimiento inductivo de un programa de computadora que produzca alguna salida deseada cuando se le presenta alguna entrada en particular. Esto es precisamente lo que la metodología de PG realiza de una manera sistematizada.

Con base en este planteamiento un programa de computadora puede ser llamado una fórmula, un plan, una estrategia de control, un procedimiento computacional, etcétera.
Similarmente, las entradas del programa de computadora pueden ser llamadas variables independientes, variables de estado, valores de sensores, argumentos de función, etcétera. A su vez las salidas del programa de computadora pueden denominarse variables dependientes, un movimiento, un actuador, el valor regresado por una función, etcétera.

Capítulo III

Sin importar la diferencia en terminología, las razones para reformular los problemas planteados por la inducción de programas como la búsqueda de un programa de computadora, son tres:

1. Los programas de computadora tienen la flexibilidad necesaria para expresar soluciones a una amplia variedad de problemas.

2. Los programas de computadora pueden tomar el tamaño, la forma y la complejidad estructural necesarios para resolver estos problemas.

3. La Programación Genética proporciona una metodología para realizar inducción de programas.

Con estos puntos, se observa que los programas pueden ser el lenguaje para expresar varios problemas.

ÁREA DEL PROBLEMA	PROGRAMA DE COMPUTADORA	ENTRADA	SALIDA
Control óptimo	Estrategia de control	Variables de Estado	Variable de Control
Planeación	Plan	Valores de sensores o desconectores	Acciones de actuadores
Diseño de filtros adaptivos	Filtro FIR O IIR	Señal de datos y señal de ruido	Señal de datos
Descubrimiento de identidades matemáticas	Nuevas expresiones matemáticas	Muestra aleatoria de valores de las variables independientes de la expresión matemática diaria	Valores de la expresión matemática dada
Programación Automática de un autómata celular	Reglas de transición de estado para la célula	Estado de la célula y sus vecinos	Próximo Estado de la Célula

TABLA III.1: Algunas áreas de problemas que pueden ser expresados como un problema de inducción de programas.

En la tabla III.1 se presentan diversas áreas que pueden ser expresadas como un problema de inducción de programas.

Esta tabla usa la terminología empleada para describir las entradas, salidas y el programa de computadora de diferentes áreas de problemas, desde la perspectiva de un problema de inducción de programas.

Se observa que diferentes problemas pueden ser reformulados de esta manera.

Por ejemplo, el control óptimo consiste en encontrar una estrategia (de control), representada por un programa de computadora, que use las variables del estado actual de un sistema para escoger un valor en las variables de control que causen que el estado del sistema se mueva hacia la meta deseada, mientras se maximiza o minimiza alguna medición de costo.

Un ejemplo de un problema de control óptimo simple involucra descubrir una estrategia de control para centrar un carro en una pista en un tiempo mínimo. En este ejemplo, las variables de estado del sistema son la posición y la velocidad del carro. La estrategia de control especifica cómo escoger la fuerza que es aplicada al carro, pues la aplicación de la fuerza causa que el estado del sistema cambie.

El estado destino deseado es que el carro descanse en el punto central de la pista.

El siguiente diagrama de flujo muestra el comportamiento de la PG. El índice i, se refiere a un individuo en la población M. La variable GEN representa el número de generación actual.

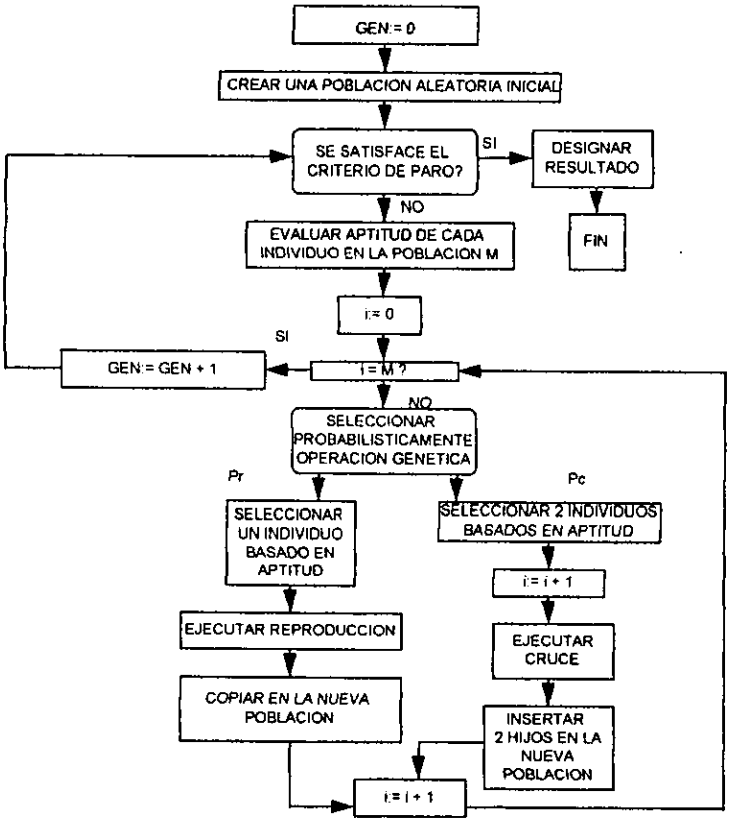


FIGURA III.1: DIAGRAMA DE FLUJO DEL PARADIGMA DE LA PROGRAMACIÓN GENÉTICA

III.4 Descripción de la programación genética

En esta sección se describe la técnica de la Programación Genética, de manera detallada.

La adaptación (o aprendizaje) involucra el cambio de alguna estructura, así ésta presenta una mejor adaptación al medio. La Programación Genética se describe en los siguientes términos:

- Estructuras adaptivas,
- El conjunto de estructuras iniciales o funciones primitivas,
- La medición de la Aptitud,
- Las operaciones genéticas,
- El estado del sistema,
- El criterio para terminar la ejecución,
- El método para designar el resultado final y
- Los parámetros de control para el proceso.

Primeramente se definirá la forma en que se mapea el espacio de soluciones utilizando los conjuntos de terminales y funciones primitivas; enseguida se tratan las operaciones que controlan la evolución de los programas, la medición de la Aptitud y algunas características propias de la Programación Genética.

III.4.1 Estructuras adaptivas

Cada sistema evolutivo o adaptivo, en la última estructura sufre una adaptación. Para los algoritmos genéticos convencionales y la programación genética, las estructuras que se adaptan son una población de puntos individuales del espacio de búsqueda, más bien, que un punto individual. Los métodos genéticos difieren de otras técnicas de búsqueda en las que simultáneamente se involucra una búsqueda paralela de cientos o miles de puntos en el espacio de búsqueda.

Las estructuras individuales que sufren una adaptación, en programación genética, son estructuradas jerárquicamente en programas de computadora. El tamaño, la forma y el contenido de estos programas pueden cambiar dinámicamente durante el proceso.

El conjunto de estructuras posibles en programación genética, es el conjunto de todas las posibles composiciones de funciones que es posible formar recursivamente del conjunto de funciones N_{func} de F = {f1, f2, ...fN_{func}} y el conjunto de terminales N_{term} de T = {a1, a2, ..., aN_{term}}. Cada función particular f1 en el conjunto F toma un número específico z(fi) de argumentos z(f1), z(f2),...,z(fN_{func}). Esto es, la función fi tiene z(fi) argumentos.

Las funciones de el conjunto de funciones deben incluir:

- operadores aritméticos (+, -, *, etc.),
- funciones matemáticas (sen, cos, exp y log),
- operadores booleanos (AND, OR, NOT),

- operadores condicionales (if - then - else),
- funciones iterativas (do - until),
- funciones recursivas,
- y otras funciones con dominio previamente especificado.

Los terminales, generalmente son variables átomo (salidas, sensores, detectores o estados variables de algún sistema) o constantes (tales como el número 3 o el constante booleano NIL). Ocasionalmente, los terminales son funciones que toman argumentos no explícitos, la funcionalidad real de tales funciones falsean los efectos colaterales en el estado del sistema.

Considere el conjunto de funciones: F = {AND, OR NOT} y el conjunto terminal: T = {D0, D1}, donde D0 y D1 son variables booleanas que sirven como argumentos de las funciones. Se puede combinar el conjunto de funciones y terminales en un conjunto C como el siguiente:

$$C = F \cup T = \{AND, OR, NOT, D0, D1\}$$

Entonces, se puede ver que los terminales en el conjunto C como funciones requieren argumentos de valor cero. Esto es, los 5 elementos de C, pueden ser vistos como 2, 2, 1, 0 y argumento 0 respectivamente.

Como ejemplo considere la función de paridad 2 (i.e., NOT exclusivo) con 2 argumentos. Esta función devuelve T(verdadero) si un número par de estos argumentos (i.e., D0, D1) son verdaderos, de otro modo, devuelve NIL(falso). Esta función booleana puede expresarse en forma normal disjuntiva (DNF) para la siguiente expresión-S LISP:

(OR(AND(NOT D0) (NOT D1)) (AND D0 D1))

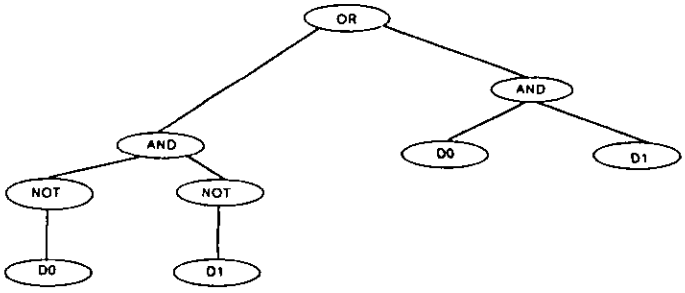


FIGURA III.2: Función de paridad 2.

La figura III.2 describe gráficamente la anterior expresión-S LISP. Formada por una raíz y un árbol. Los cinco puntos internos del árbol están etiquetados con funciones (OR, AND, NOT, NOT y AND). Los cuatro puntos externos se etiquetan con terminales (variables booleanas D0 y D1, respectivamente).

La raíz corresponde a la función que aparece justo en el extremo izquierdo del paréntesis de la expresión (OR). Este árbol es equivalente al árbol analizador que los compiladores construyen internamente al ser representado un programa dado.

Capítulo III

El espacio de búsqueda para la programación genética en el espacio de mapeo de todas las expresiones-S LISP que pueden ser recursivamente creadas por composiciones equivalentes disponibles, puede ser visto como el espacio de árboles ramificados ordenadamente teniendo como puntos internos funciones disponibles y puntos externos con terminales disponibles.

Las estructuras que sufren adaptación en PG, son diferentes de las estructuras que se adaptan en un algoritmo genético convencional operando con cadenas. En PG, son estructuras jerárquicas. Las estructuras que sufren adaptación en un algoritmo genético son cadenas unidimensionales de longitud lineal fija.

En PG, el conjunto de terminales y el conjunto de funciones seleccionado debe satisfacer la propiedad de cerradura y suficiencia.

III.4.1.1 Propiedad de cerradura para el conjunto de funciones y el conjunto de terminales

La propiedad de cerradura requiere cumplirse por cada función del conjunto de funciones, los argumentos asignados y los argumentos que posiblemente se devuelvan por una función, así como los valores que sean asumidos y devueltos por el conjunto de terminales. Esto es, cada función en el conjunto de funciones debe ser bien definida y alguna combinación de argumentos que sea encontrada debe garantizar la propiedad de cerradura.

En el caso simple donde el conjunto de funciones consiste de funciones booleanas, tales como AND, OR y NOT y el conjunto de terminales consiste de variables booleanas que puedan asumir únicamente los valores de T o NIL, se satisface fácilmente la propiedad de cerradura. Sin embargo, los programas de computadora comunes, contienen variables numéricas, operadores condicionales y de bifurcación.

En programas ordinarios, algunas veces las operaciones aritméticas con variables numéricas están indefinidas (i.e., división por cero). Muchas funciones matemáticas comunes con variables numéricas, en ocasiones, son tipos de datos inaceptables en un programa particular (raíz cuadrada o logaritmo de un numero negativo). Los valores booleanos (T o NIL) resultado típico de un operador condicional, generalmente, no son aceptados como argumentos en una operación aritmética.

La propiedad de cerradura no siempre puede ser satisfecha por programas ordinarios, o si es posible cumplirla, se llama a una compleja y restrictiva estructura sintáctica impuesta en el programa.

Si la división aritmética encuentra el valor de cero como segundo argumento, la propiedad de cerradura no será satisfecha, al menos que se haga alguna disposición antes con la posibilidad de división por cero.

Se debe definir un método simple, que garantice la cerradura, para proteger la función de división. Esta protección, toma el porcentaje de 2 argumentos y regresa 1 cuando la división por cero es tentativa (incluyendo 0 dividido por 0) y en otro caso, regresa un cociente normal.

Esto puede codificarse en LISP[5] como:

```
(defun % (numerator denominator)
"The Protected Division Function"
(if(= 0 denominator) 1 (/ numerator denominator)))
```

Alternativamente, se logra la propiedad de cerradura por definición de la división, así como el valor resultante :undefined y entonces se reescribe cada una de las funciones aritméticas como el valor resultante :undefined siempre que se encuentre :undefined como uno de sus argumentos.

[5] LISP: *Lenguaje de procesamiento de listas algebraicas, proporciona recursividad y funciones que lo hacen muy popular en aplicaciones de inteligencia artificial y gracias a sus características de estructura, resulta de fácil manejo y mejor comprensión para los usuarios de programación genética.*

Si la raíz cuadrada encuentra un argumento negativo o si la función logarítmica encuentra un argumento no positivo, en un problema donde el número complejo que comúnmente debe resultar es inaceptable, se puede garantizar la cerradura usando una función de protección. Por ejemplo, la función de protección para la raíz cuadrada SRT toma un argumento y regresa el valor absoluto de ese argumento. Esto puede codificarse como sigue:

```
(defun srt (argument)
"The Protected Square Root Function"
(sqrt (abs argument))),
```

donde SQRT es la función de raíz cuadrada en Common LISP.

La protección para logaritmos naturales RLOG devuelve el valor 0 si un argumento es 0, de otro modo, devuelve el logaritmo natural del valor absoluto de ese argumento. Esto puede codificarse en LISP como:

```
(defun rlog (argument)
"The Protected Natural Logarithm Function"
(if (= 0 argument) 0 (log (abs argument)))),
```

donde LOG es la función de algoritmo natural en COMMON LISP.

Si un programa contiene un operador condicional en un problema donde el valor booleano que resulte es inaceptable, entonces el operador condicional puede ser modificado con alguno de los siguiente tres casos:

- Puede usarse un valor numérico.
- Pueden redefinirse los operadores condicionales comparativos.
- Pueden redefinirse los operadores condicionales de bifurcación.

La propiedad de cerradura es deseable. pero no absolutamente necesaria. Si la cerradura no prevalece, se tienen alternativas, tales como descartar individuos evaluados con resultados inaceptables o asignar algún castigo o pena a individuos no factibles.

Note que la propiedad de cerradura se requiere únicamente por terminales y funciones que se puedan generar. Si las estructuras que sufren adaptación se sabe que cumplen con reglas de construcción sintáctica, la cerradura sólo se requiere sobre los valores de terminales y los valores resultantes por las funciones que sean encontradas.

III.4.1.2 Suficiencia del conjunto de funciones y el conjunto de terminales

La propiedad de suficiencia requiere que el conjunto de terminales y el conjunto de funciones primitivas o iniciales, sea capaz de expresar una solución al problema. El usuario de PG debe conocer alguna composición de las funciones y terminales que sean aplicables al campo solución del problema.

La identificación de las variables que tienen el suficiente poder explicativo para resolver un problema particular es un problema en la ciencia.

Dependiendo del problema, este paso puede ser obvio o tal vez requiera un concienzudo estudio. En algunos campos, la tarea de identificar las variables que pueden resolver un problema, puede ser imposible (i.e., rangos de producción o resultados de la elección). Similarmente, la identificación de variables con criterio de suficiencia para un conjunto de funciones que resuelvan un problema en particular puede ser obvia o muy compleja.

Nada de la Teoria de Autómatas, Redes Neuronales, Algoritmos Genéticos, Máquinas de Conocimiento o Inteligencia Artificial, provee alguna asistencia para seleccionar un conjunto de funciones primitivas para estos problemas o establecer un conjunto particular de funciones primitivas que provean un criterio de suficiencia.

III.4.1.3 Universalidad del conjunto de funciones y el conjunto de terminales

Los pasos (presentados por el usuario) de determinación del repertorio de funciones primitivas y terminales en PG, son equivalentes a simular los pasos requeridos en otros paradigmas de máquinas de conocimiento. Estos pasos, muchas veces no son identificados explícitamente, discutidos o reconocidos por investigadores que describen otros paradigmas. La razón para ésta omisión, tal vez se deba a que los investigadores involucrados consideren que la elección de funciones primitivas y terminales sea inherente a la declaración del problema. Este punto de vista es entendible si el investigador está enfocado a un único tipo de problema de un campo específico.

Los pasos de determinación de funciones primitivas y terminales son necesarios para preparar la solución a un problema usando algoritmos de inducción de árboles de decisión (tales como ID3), un algoritmo empírico (como BACON), una red neural, un autómata de estado finito, un sistema clasificador genético y otros paradigmas. En cada caso, el usuario debe identificar y suplir las funciones primitivas y terminales a ser usadas en la solución del problema.

Al escoger un conjunto de funciones y terminales, por supuesto, se afecta directamente el carácter y apariencia de las soluciones. Las funciones y terminales disponibles forman la base del potencial de soluciones que puedan generar.

Por supuesto, en algunos problemas, el conjunto de funciones no es muy claro, así que es mínimamente suficiente para resolver un problema. En estos casos, generalmente, se incluye una función artificial para encontrar una solución.

El efecto del funcionamiento de los terminales artificiales es más claro que el de las funciones. Generalmente, los terminales artificiales reducen la ejecución del algoritmo.

III.4.2 Estructuras iniciales

Las estructuras iniciales en PG consisten de individuos de la población inicial de una expresión-S para el problema. La generación de cada expresión-S en la población inicial es hecha de forma aleatoria en una raiz de un árbol ramificado que represente dicha expresión.

Si un terminal se escoge para ser la etiqueta de algún punto, éste punto comienza en un punto extremo del árbol y el proceso generado se finaliza por dicho punto.

Por ejemplo, en la figura III.3, el terminal A del conjunto de terminales T se selecciona para ser la etiqueta de la primera línea que se desprende del punto etiquetado con la función *. Este proceso continúa de izquierda a derecha hasta que el árbol queda completamente etiquetado.

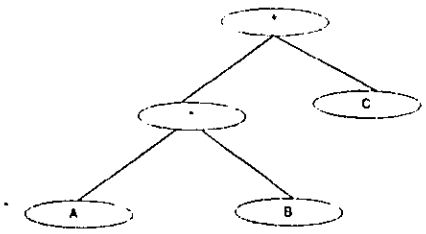


Figura III.3: Creación de un árbol con terminales A, B y C.

Este proceso de generación, puede implementarse de muchas maneras, resultando árboles aleatorios de distintas formas y tamaños. La profundidad o extensión de un árbol se define como la longitud de la trayectoria de la raiz a un punto final.

En la PG, los individuos duplicados aleatoriamente son especialmente parecidos a los creados en la generación aleatoria inicial cuando los árboles son pequeños. Cada expresión-S es revisada por singularidad antes de ser insertada en la población inicial. Si una nueva expresión-S es un duplicado, el proceso generado es repetido hasta que una expresión-S singular se crea.

La variedad de una población es el porcentaje de individuos para los cuales no existen duplicados exactos en la población.

Si se duplica la verificación, la variedad de una población inicial aleatoria es 100%. En generaciones tardías, la creación de individuos duplicados vía operación genética de reproducción es una parte inherente del proceso genético. En contraste, en un algoritmo genético convencional, la operación de cadenas de caracteres de longitud fija, cada carácter en una cadena en la población inicial aleatoria, se crea típicamente, llamando un aleatorio binario.

Si cada preparación es ensayada, debe recordarse que insertar individuos de relativa adaptabilidad, dentro de una población inicial de individuos aleatorios los cuales después de una generación, resultan en total dominio de la población por copias y descendientes de los individuos principales. En términos de diversidad genética, el resultado será, después únicamente una generación, muy similar a comenzar con una población de tamaño igual al número relativamente débil de individuos principales. Si cada prueba es ensayada, 100% de la población inicial debe ser probada con individuos de un nivel similar de adaptación.

III.4.3 Medición de la Aptitud

La aptitud es el principio de la selección natural de Darwin y en su caso, para los algoritmos genéticos y la programación genética.

En la naturaleza, la aptitud de un individuo es la probabilidad de sobrevivir o adaptarse al medio. En el mundo artificial de los algoritmos matemáticos, se mide la aptitud de alguna forma y se usa como control a la aplicación de las operaciones que modifican las estructuras en una población artificial.

La aptitud se mide de muchas maneras, algunas explícitas y otras implícitas.

El método más común para medir la aptitud, es crear una función explícita de aptitud para cada individuo en la población. Cada individuo en una población se asigna a un valor escalar de aptitud que significa definir bien algunos procedimientos explícitos de evaluación.

El hecho de la existencia de los individuos y sobrevivencia en la población y su posible reproducción debe ser indicada por la aptitud o capacidad de adaptación (como en la naturaleza).

Existen cuatro medidas para la función de aptitud:

- Aptitud inicial
- Estandarización
- Ajuste
- Normalización

A continuación se describen brevemente éstas medidas.

III.4.3.1 Aptitud inicial

La aptitud inicial es la medida de él estado natural en términos del problema mismo.

La aptitud es, generalmente, pero no siempre, evaluada sobre un conjunto de casos de aptitud. Estos casos proveen una base para la evaluación de la aptitud en expresiones-S en la población sobre un número de diferentes situaciones representativas, suficientemente grande en un rango numérico de diferentes valores de aptitud inicial que se obtengan. Los casos de aptitud, usualmente, son solamente un pequeño espacio finito de enteros. Para las funciones booleanas, es muy práctico, el uso de todas las posibles combinaciones de valores de los argumentos como casos de aptitud. Estos casos deben ser representativos del espacio, como un entero, porque forman la base para la generalización de los resultados obtenidos en el espacio del dominio de los enteros.

Se puede minimizar el efecto de seleccionar un conjunto particular de casos de aptitud por el programa de computadora, usando un conjunto diferente de casos de aptitud en cada generación.

Los casos de aptitud, se escogen en el comienzo de cada corrida y no varía de generación en generación.

El término más común para referirse a la aptitud inicial es el de error inicial. Esto es, la aptitud inicial de una expresión-S es la suma de las distancias, tomadas sobre todos los casos, entre el punto en el rango de la expresión, para el conjunto de argumentos y el punto correcto en el espacio asociado al caso particular de aptitud. La expresión-S puede ser un valor booleano, un entero, un punto flotante, un número complejo, un vector o un valor múltiple.

Si la expresión-S es un entero o un punto flotante, la suma de distancias, es la suma de los valores absolutos de las diferencias entre los valores involucrados. Cuando la aptitud inicial es error, la función r(i,t) de una expresión-S individual i, en la población de tamaño n en alguna generación t es:

$$r(i, t) = \sum_{j=1}^{Nc} | s(i, j) - c(j) |$$

donde s(i, j) es el valor resultante de i, del caso j (Nc casos) y donde c(j) es el valor correcto de aptitud para j. Si la expresión es un valor booleano, la suma de las distancias es equivalente al número de uniones. Si es un número complejo, vectorial o múltiple, la suma de las distancias es igual a la suma de las distancias obtenidas separadamente por cada componente de la estructura involucrada. Si es un valor entero o real, la raíz cuadrada de la suma de los cuadrados de las distancias pueden, alternativamente, ser una medida de aptitud inicial.

La aptitud inicial es un estado natural del problema, el mejor valor puede ser el más pequeño (cuando la aptitud se maneja como error) o muy grande (cuando se presenta como un beneficio).

III.4.3.2 Estandarización

La función de estandarización s(i, t) finaliza con la aptitud inicial. Así el valor más bajo es siempre el mejor valor. Por ejemplo, en un problema de control, una medida de costo mínimo, el valor menor de la aptitud inicial es el mejor. Similarmente, si en un problema particular, 1 es el error mínimo, un valor aún más pequeño para la aptitud inicial es mejor. Si para un problema dado, un valor de la aptitud inicial es mejor, la función de aptitud estándar es igual a la aptitud inicial del problema. Es decir:

$$s(i, t) = r(i, t)$$

Es conveniente, y también deseable, que el mejor valor de la función de estandarización sea igual a 0. Si un valor muy pequeño de la aptitud inicial es mejor, el signo se reserva y las funciones de ajuste y normalización deben programarse directamente de la aptitud inicial.

III.4.3.3 Ajuste

La función de ajuste a(i, t) se calcula de la función de estandarización s(i, t):

$$a(i, t) = 1 / 1 + s(i, t)$$

donde s(i, t) es la función estándar de un individuo i en un tiempo t.

La función de ajuste se encuentre entre 0 y 1. El valor de ajuste es el mayor de los mejores individuos en la población. La función de ajuste tiene el beneficio de exagerar la importancia de las pequeñas diferencias en los valores de la aptitud estándar, como el acercamiento a 0 de la estandarización (como en el caso de generaciones grandes). Dado que la población mejora, el énfasis mayor, es el lugar de las pequeñas diferencias que existen entre un buen individuo y uno muy bueno. Esta exageración es especialmente poderosa si la estandarización de la aptitud es 0 cuando se ha encontrado una solución perfecta al problema.

III.4.3.4 Normalización

Si el método que se emplea es el de aptitud proporcional, el concepto de normalización es necesario. La función de normalización n(i, t) es calculada del valor de la función de ajuste a (y, t) como sigue:

$$n(i, t) = a(i,t) / \sum_{k=1}^{m} a(k, t)$$

La normalización tiene 3 características:

- Rangos entre 0 y 1
- El mejor individuo de la población
- La suma de los valores de la normalización es 1

Es posible que para la función de aptitud tenga algún peso como factor secundario o terciario.

III.4.4 Las operaciones genéticas

A continuación se describen dos de las operaciones usadas en programación genética:

- Reproducción
- Cruce (o recombinación sexual)[6]

III.4.4.1 Reproducción

Para la PG, la operación de reproducción, es el principio básico al igual que en la naturaleza. La reproducción es asexual y opera únicamente en la expresión padre y únicamente produce una expresión hijo en cada ocasión que se presenta.

La reproducción consiste de dos pasos. Primero, se selecciona un individuo de la población de acuerdo a algún método basado en aptitud. Segundo, se copia el individuo, sin alteración, dentro de la nueva población (i. e., la nueva generación).

Existen muchos métodos de selección basados en aptitud. El más popular es la selección de aptitud proporcional. Este método se describe de la siguiente manera.

[6] Existen también cinco operaciones secundarias: mutación, permutación, edición, encapsulamiento y decimación.

Si f(si (t)) es la aptitud de un individuo si en la población en la generación t, entonces por aptitud proporcional, la probabilidad de que el individuo s, se copie en la siguiente generación es:

$$f(s_i(t)) / \sum_{j=1}^{m} f(s_j(t))$$

Tipicamente, f (s, (t)) es la normalización de aptitud de n(s, (t)), así la probabilidad de que el individuo s, se copie dentro de la siguiente población simplemente se normaliza $n(s_{i,} (t))$.
El padre permanece en la población mientras la selección se efectúa durante la generación presente. Esto es, la selección se hace con reemplazo. Los padres pueden ser seleccionados y generalmente, más de una vez para reproducirse durante la generación presente.

Un considerable tiempo de programación, puede salvarse no procesando la aptitud para algún individuo que aparezca en la generación actual como resultado de una operación de reproducción en la generación previa.

La aptitud de un individuo copiado será inmutable y no será necesaria ser reprocesada (a excepción de los casos de aptitud que varían de generación en generación). Si la reproducción se aplica, por decir, al 10% de la población en cada generación, esta técnica por sí sola resultará en 10% menos cálculos de aptitud en cada generación. Esta técnica produce un ahorro inmediato de tiempo de cerca del 10% en cada corrida.

III.4.4.2 Cruce

La operación de Cruce en PG produce una variación en la población por la producción de nuevos hijos que consisten de partes tomadas de cada padre. El cruce se prepara con dos padres y produce dos hijos. Es decir, es una operación sexual. El primer padre se escoge de la población por el mismo método de selección usado para la reproducción. El segundo se escoge con una probabilidad igual a la aptitud normalizada. La operación comienza por selección independiente, usando un punto aleatorio en cada padre para que sea el punto de cruce para el padre. Los dos padres son generalmente de tamaño distinto.

El fragmento de cruce para un padre, es la raíz del árbol, el cual ha sido raíz del punto de cruce para el padre (i. e., más distante de la raíz que el árbol original).

El primer hijo, se genera borrando el fragmento de cruce del primer padre y entonces se inserta el segundo fragmento del segundo padre en el punto de cruce del primer padre. El segundo hijo se genera de manera simétrica.

En la figura III.4, se muestra un ejemplo de la operación de Cruce en la que se observan los terminales (coeficientes) y las funciones primitivas (suma, resta, multiplicación y división). Cada uno de los nodos en los padres se etiqueta con un número. Se escogen los nodos 2 del padre de la izquierda y 5 del padre de la derecha como puntos de cruce y se producen los árboles-hijo intercambiando estos nodos. El espacio de búsqueda en la PG es el espacio de todos los posibles programas de computadora compuestos de funciones y terminales apropiados al dominio del problema.

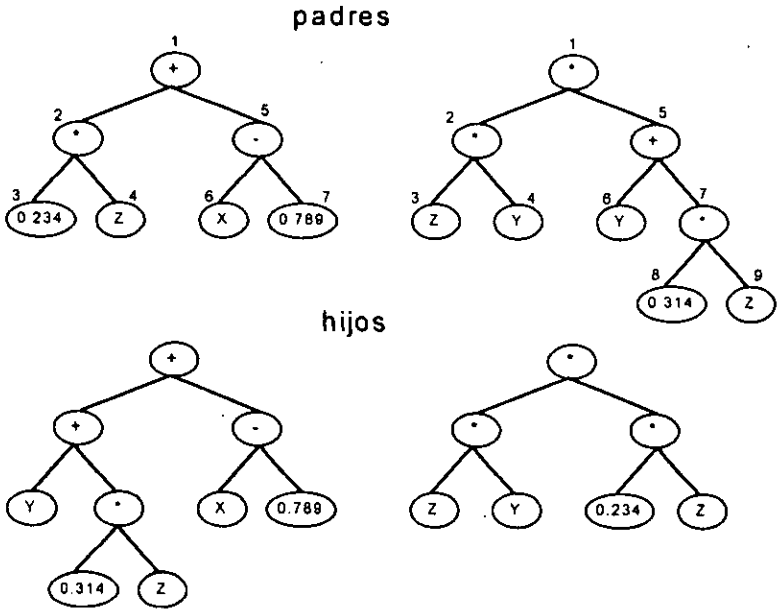


FIGURA III.4: Ejemplo de cruce

El tamaño máximo permisible (medida de densidad del árbol) se establece para los hijos creados por la operación de cruce. Este límite, prevé el desgaste de largas horas de tiempo máquina en un par de expresiones individuales extremadamente grandes. Si un cruce de dos padres genera un hijo de tamaño no permitido la función de cruce se aborta, y el primero de los padres se escoge arbitrariamente para ser reproducido dentro de la nueva población. Los otros hijos deben ser de tamaño permisible.

III.4.5 Estado del sistema

En la PG, el estado del sistema (memoria), en algún punto durante el proceso, consiste únicamente de la población presente o actual. No es necesario memoria adicional o contabilidad centralizada.

En una implementación de computadora del paradigma de la programación genética, es necesario tener parámetros de control para la corrida: el conjunto de terminales y el conjunto de funciones primitivas.

III.4.6 Criterio de paro

El criterio de paro para la programación genética es el número máximo establecido previamente de generaciones G que deben ser procesadas.

III.4.7 Designación de Resultados

El método para designar resultados para la PG, es el de designar el mejor individuo que aparece en una generación (i.e., el individuo más apto) como el resultado de la corrida de un programa genético. No se garantiza que el mejor individuo permanezca en todas las generaciones subsecuentes.

Un método alternativo es designar la mejor generación en la población en el tiempo de terminar como el resultado de una corrida. Este método, usualmente produce el mismo resultado que el anterior, porque el mejor individuo, generalmente, se encuentra en la población al tiempo de terminar (i. e., se encuentra en la mejor generación de la generación anterior).

Las razones para uno u otro, son :

a) éste fue creado en una generación temprana y es altamente apto o fue copiado dentro de la generación actual por la operación de reproducción.

b) La corrida termina con la generación presente por virtud de la creación de varios individuos (i. e., satisfacen el criterio de paro).

En algunos problemas, la población en conjunto o una subpoblación seleccionada proporciona la aptitud que es designada como el resultado. En este caso, el conjunto de individuos actuales, se presenta como un conjunto de soluciones alternativas a el problema(i. e., una estrategia mixta).

III.4.8 Parámetros de control

El paradigma de la PG, está controlado por 19 parámetros de control, incluyendo 2 mayores que son numéricos, 11 menores y 6 variables cuanlitativas; a esta selección se suman varias maneras alternativas de ejecutar una corrida.

Los dos parámetros principales o mayores son el tamaño de la población (M) y el número de generación a ser procesada (G).

- La población de tamaño M es 500
- El número máximo de generaciones G es 51 (una generación aleatoria inicial, llamada generación 0, más 50 generaciones subsecuentes)

Los once parámetros menores para controlar el proceso son:

- La probabilidad de cruce, pc, es 0.90. Esto es, el cruce se presenta en 90% de la población para cada generación. Por ejemplo, si el tamaño de la población es 500, entonces 450 individuos (225 pares) de cada generación se seleccionan (con reselección permisible) a participar en el cruce.
- La probabilidad de reproducción, pr, es 0.10. Por ejemplo, si la población es de 500, 50 individuos de cada generación se seleccionan para reproducirse.
- En la selección de los puntos de cruce, se usa una distribución de probabilidad p_p = 90% de los puntos de cruce, equivalentemente también a los puntos internos (funciones) de cada árbol y 10% de los puntos de cruce también iguales a los puntos externos (terminales) de cada árbol.
- Un tamaño máximo, $D_{created}$, se establece 17 para las expresiones creadas por la operación de cruce.
- Un tamaño máximo, $D_{initial}$, se establece como 6 para los individuos aleatorios generados por la población inicial.
- La probabilidad de mutación, p_m, específicamente la frecuencia de la mutación es 0.
- La probabilidad de permutación, p_p, especifica la frecuencia de permutación igual a 0.
- El parámetro de frecuencia, f_{ed}, aplicado a la operación de edición es 0.
- La probabilidad de encapsulación, p_{en}, es igual a 0.
- La condición para invocar la operación de mortandad en NIL.

- El porcentaje de mortandad, pd, es irrelevante si la condición para invocar la operación de mortandad es NIL y el conjunto arbitrario es 0.

Las siguientes 6 variables cualitativas seleccionan maneras distintas de ejecutar una corrida:

- El método generativo para la población inicial aleatoria se mapea mitad y mitad.
- El método de selección para reproducción y para el primer padre en el cruce es reproducido por aptitud proporcional.
- El método de selección para el segundo padre para un cruce, es el mismo que para el primero.
- Se usa como medida la función de ajuste de la aptitud.
- La sobreselección no se usa para poblaciones de 500 y menores, y se usa para poblaciones de 1000 y superiores.
- La estrategia del elitismo no se usa.

III.4 9 Características

De acuerdo a lo expuesto en este capitulo, se pueden resumir en los siguientes renglones las caracteristicas de la PG, cuando se usa para la solución de algún problema:

1. Generar una población inicial de composiciones aleatorias de las funciones y terminales de el problema (programas de computadora).
2. Ejecutar iterativamente los siguientes pasos hasta que el criterio de paro sea satisfecho:

- Ejecutar cada programa en la población y asignar un valor de aptitud de acuerdo a que tan bien resuelve el problema.
- Crear una nueva población de programas aplicando las dos operaciones primarias. Las operaciones se aplican al programa(s) en la población que se escoge con probabilidad basada en aptitud.

  i) Copiar los programas existentes a la nueva población

  ii) Crear nuevos programas por recombinación genética escogiendo aleatoriamente partes de 2 programas existentes.

3. El mejor programa que aparezca en alguna generación se designa como el resultado de la PG. Este resultado será la solución o una solución aproximada al problema.

III.5 Ejemplo de uso

Para ilustrar la aplicación de la Programación Genética, se expone el siguiente ejemplo: REGRESIÓN SIMBÓLICA:

La regresión simbólica (i.e., una función de identificación) involucra encontrar una expresión matemática, en forma simbólica, que este provista de un buen, mejor, o perfecto punto de aptitud entre una muestra finita de valores de las variables independientes y los valores asociados a las variables dependientes. Esto es, la regresión simbólica involucra encontrar un modelo apto a una muestra de datos dado.

Cuando las variables son valores reales, la regresión simbólica requiere encontrar la forma de la función y los coeficientes numéricos para el modelo. La regresión simbólica, difiere de la lineal, cuadrática o polinomial, las cuales únicamente implican encontrar los coeficientes numéricos para una función con esa forma particular (lineal, cuadrática o polinomial).

En algunos casos, las expresiones matemáticas que son el inicio en regresión matemática, pueden verse como un programa de computadora que toma los valores de las variables independientes como entrada y devuelva como salida los valores de las variables independientes.

En el caso de datos del mundo real, el problema de encontrar el modelo de los datos se llama, frecuentemente, desarrollo empírico. Si los rangos de la variable independiente están en los enteros no negativos, la regresión simbólica es llamada inducción de secuencia.

Suponer que se tiene una muestra de valores numéricos de un punto en la curva sobre 20 puntos en el mismo dominio en el intervalo de números reales [-1.0, +1.0]. Esto es, se obtiene una muestra de datos en forma de 20 pares (x_i, y_i), donde x_i, es un valor de la variable independiente en el intervalo y y_i , es el valor asociado a la variable dependiente. Los 20 valores de x_i se eligieron de forma aleatoria en el intervalo. Por ejemplo, estos 20 pares (x_i, y_i) podrían incluir pares como: (-0.40, -0.2784), (+0.25, +0.3320)..., y (+0.50, +0.9375).
Estos 20 pares son los casos de aptitud que podrían ser usados al evaluar la aptitud de alguna función propuesta.

El objetivo es encontrar una función en forma simbólica, que sea adecuada para los 20 pares de datos. La solución al problema de encontrar una función adecuada a la muestra obtenida de datos se debe ver como una expresión matemática (expresión-S) de un espacio de posibles expresiones-S que se pueden formar de un conjunto de funciones y terminales.

El primer paso para usar PG es identificar el conjunto de terminales. En éste caso, la información que la expresión matemática debe procesar es el valor de la variable independiente X. Así, el conjunto de terminales es: T={X}.

El segundo paso, consiste en determinar el conjunto de funciones que será usado al generar la función de los datos de prueba. Entonces el conjunto de funciones F para el problema, se integra por 8 funciones (6 de las cuales son extrañas, al problema inmediato) y es:

$$F = \{ +, -, *, \%, \sin, \cos, EXP, RLOG)$$

Tomando 2, 2, 2, 2, 1, 1, 1 y un argumento respectivamente

El tercer paso, es determinar la medida de aptitud. La aptitud inicial para este problema es la suma, tomada sobre los 20 casos de aptitud, de el valor absoluto de las diferencias (error) entre los valores del espacio del rango real producido por la expresión-S para un valor dado de la variable independiente. x_i y el y_i correcto dentro del rango. Este valor debe ser 0 o cercano a 0. La estandarización de la aptitud es igual a la aptitud inicial del problema.

Los puntos óptimos, para éste problema, cuentan con el número de casos de aptitud para los cuales el valor numérico devuelto por la expresión-S se encuentra dentro de un pequeño margen de tolerancia (llamado criterio óptimo) del valor correcto.

La tabla siguiente muestra las características del problema de regresión simbólica para $x^4 + x^3 + x^2 + x$.

Objetivo:	Encontrar una función de una variable independiente y una dependiente, en forma simbólica, que se encuentre en la muestra de 20 puntos (x_i, y_i), donde la función objetivo es el polinomio de grado 4: $x^4+x^3+x^2+x$.
Terminales:	X (*la variable independiente*)
Conjunto de Funciones:	+, -, *, %, sin, cos, EXP, RLOG.
Casos de Aptitud:	La muestra de 20 puntos (x_i, y_i) donde x_i viene del intervalo [-1, +1].
Aptitud inicial:	La suma, tomada de los 20 casos de aptitud, de el valor absoluto de la diferencia entre el valor de la variable dependiente obtenida por la expresión-S y el valor objetivo de y_i *de la variable dependiente*.
Estandarización de la Aptitud:	Igual que para la aptitud inicial para éste ejemplo.
Óptimos:	Número de casos de aptitud para los cuales el valor de la variable dependiente producida por la expresión-S se encuentra dentro de 0.01 y el valor de y_i.
Parámetros:	M = 500 y G = 51.
Predicado de éxito:	Una expresión-S con 20 óptimos.

TABLA III.2: TABLA PARA EL PROBLEMA DE REGRESIÓN SIMBÓLICA.

La población inicial aleatoria de expresiones-S incluye una gran variedad de expresiones no aptas. En una corrida, la peor generación de individuos en la generación 0 fue la expresión-S:

(EXP(-(% X (- X (sin x))) (RLOG (RLOG (* X X)) X))).

La suma de los valores absolutos de las diferencias entre la peor generación individual y los 20 puntos (i.e., la aptitud inicial) fue alrededor de 10^{38}.

Los individuos promedio en la población inicial fueron la expresión-S:

(cos (cos (+ (- (* X X) (% X X)) X)))

lo cual equivale a: cos [cos (x^2 + x - 1)]

El valor absoluto de la diferencia entre los individuos promedio y los 20 puntos fue de 23.67.

La figura III.5, muestra una gráfica de la mediana individual en el intervalo [-1,+1] de la generación 0 y la curva $x^4+x^3+x^2+x$. La distancia entre ambos valores promedia cerca de 1.2 unidades sobre los 20 casos de aptitud.

La curva no es muy cercana al punto óptimo, ésta distancia es más cercana que 10^{38}.

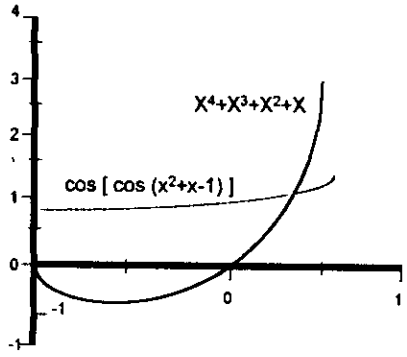


FIGURA III.5: PROMEDIO INDIVIDUAL DE LA GENERACIÓN 0 COMPARADO CON EL POLINOMIO OBJETIVO.

El segundo mejor individuo, en la población inicial fue:

$$X + [\,RLOG\ 2X + X\,] * [\,\sin 2X + \sin X^2\,].$$

El valor absoluto de la diferencia entre el segundo mejor individuo sobre los 20 casos de aptitud fue 6.05. Esto es, la aptitud inicial fue 6.05.

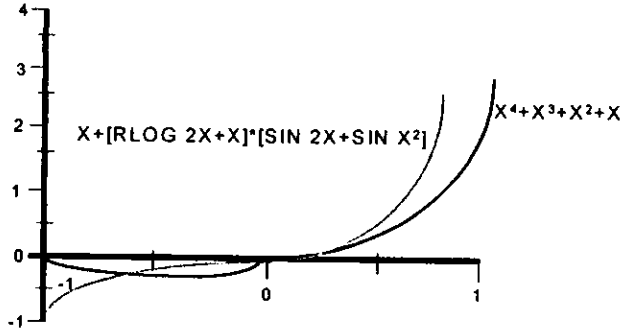


FIGURA III.6: SEGUNDO MEJOR INDIVIDUO PARA LA GENERACIÓN 0 Y EL POLINOMIO OBJETIVO.

La figura III.6 muestra la curva para el segundo mejor individuo y la curva objetivo. Ésta curva, es considerablemente más cercana a la curva objetivo, que la generada por los individuos promedio. El valor absoluto de la diferencia entre ambas curvas, es de 0.3 por caso de aptitud.

Los mejores individuos en la población de la generación 0 fue:

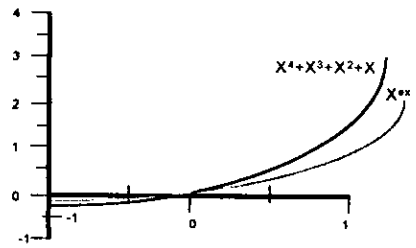
(* X (+ (+ (- (% X X) (% X X)) (sin (- X X))) (RLOG (EXP (EXP X)))))

Ésta expresión-S equivale a xe^x.

La aptitud inicial para la mejor generación fue de 4.47.

La figura III.7 muestra las curvas generadas por el mejor individuo de la generación y el polinomio, cuya diferencia promedio es alrededor de 0.22 por caso de aptitud.

FIGURA III.7: MEJOR GENERACÓN INDIVIDUAL DE LA GENERACIÓN 0 Y EL POLINOMIO OBJETIVO



La función xe^x , produjo 2 puntos óptimos para los 20 casos de aptitud, por lo tanto, no es muy conveniente o apta. Cuando se grafica xe^x , se aproxima visiblemente al polinomio $x^4+x^3+x^2+x$, primero, ambas funciones son 0 cuando x = 0.

El punto exacto de convergencia de ambas curvas a el origen, se estima para uno de los puntos óptimos para xe^x y para la curva más cercana para un valor de x cercano a 0 que se estime para el segundo punto óptimo. Segundo, cuando x se acerca a +1.0, xe^x se aproxima a 2.7; mientras que el polinomio se acerca a un valor cercano a 4.0. Así, cuando x está entre 0.0 y -0.7, xe^x y el polinomio se aproximan visiblemente.

La tabla III.3, muestra de manera simplificada, los resultados que se han obtenido en el proceso. Se muestran únicamente cinco puntos x_i del intervalo [-1,+1]. Estos 5 valores de x_i se muestran en el renglón 1 de la tabla.

El renglón 2, muestra el valor de la mejor generación $Y= xe^x$ de la generación 0 para los 5 valores de x_i

El renglón 3, contiene el conjunto de datos T representando la curva $x^4+x^3+x^2+x$. El renglón 4, representa el valor absoluto de la diferencia entre los datos T y el valor de la mejor generación $Y= xe^x$ para la generación 0. La suma de los cinco valores en el renglón 4 (i.e., la aptitud inicial) es 1.702. Si la aptitud inicial fuera 0, la función $Y= xe^x$, en el renglón 2 debería ser perfectamente apta para el conjunto de datos contenido en el renglón 3.

1	x_i	-1.0	-0 5	.00	+.5	+1.0
2	$Y= xe^x$	-.368	-.303	.000	.824	2 718
3	T	0 0	-.312	.000	.938	4.0
4	|T-Y|	.368	.009	.000	.113	1.212

TABLA III.3: REPRESENTACIÓN SIMPLIFICADA CON 5 CASOS DE APTITUD PARA EL PROBLEMA DE REGRESIÓN SIMBÓLICA.

Para la generación 2, la mejor generación fue:

(+ (* (* (+ X (* X (* X (% (% X X) (+ X X))))) (+ X (* X X))) X) X),

lo cual equivale a:

$$x^4 + 1.5x^3 + 0.5x^2 + x$$

La aptitud inicial de ésta generación fue de 2.57. Esto es, un promedio de 0.13 por caso de aptitud.

Ésta generación marcó 5 puntos óptimos en comparación a la generación 0.

Es decir, son más cercanos a la función objetivo, es un polinomio y también es un polinomio de grado 4. Además, los coeficientes también son correctos.

Antes de continuar, se debe notar que los coeficientes no numéricos fueron explícitamente definidos en el conjunto de terminales, la PG automáticamente, creó el coeficiente racional 0.5 para el término cuádratico x^2 usando primero 1/2x(por división x/x=1 multiplicado por x+x=2x) multiplicado por x. El coeficiente 1.5 para x^3 fue creado de manera semejante.

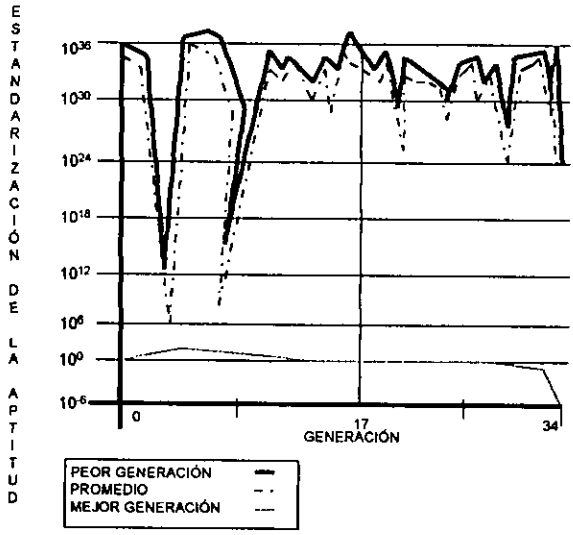


FIGURA III.8: FUNCIONES APTAS PARA EL PROBLEMA DE REGRESIÓN SIMPLE SIMBÓLICA.

La figura III.8, muestra las gráficas de los valores por generación de la estandarización de la aptitud de los mejores individuos, la peor generación y el promedio individual en la población entre las generaciones 0 y 34 resultado de las corridas del programa. Debido a las magnitudes de los valores de la estandarización de la aptitud, se usa una escala logaritimica en el eje de las ordenadas. Como puede verse, la estandarización de la aptitud de la mejor generación generalmente, mejora (i.e., disminuye) y tiende hacia la línea horizontal que representa el valor de 10^{-6} cercano a cero.

Para la generación 34, el valor absoluto de la diferencia entre los mejores individuos y el polinomio $x^4 + x^3 + x^2 + x$ sobre los 20 casos de aptitud alcanza el valor de 0.0, la primera vez que se ejecuta. Por supuesto, estos individuos también marcan 20 puntos óptimos. Estos mejores individuos están representados por la expresión-S de 20 puntos:

(+X (* (+X (* (*(+X (- cos (- X X)) (- X X))) X) X)) X))

Note que el término (- cos (- X X)) evalúa sólo a 1.0. Toda ésta expresión-S es equivalente a: $x^4 + x^3 + x^2 + x$, polinomio que representa a la función objetivo.

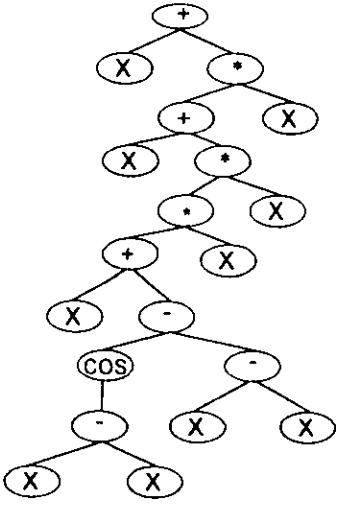


FIGURA III.9: REPRESENTACIÓN DEL 100% DE LOS INDIVIDUOS MAS APTOS GENERADOS PARA EL PROBLEMA DE REGRESIÓN SIMBÓLICA.

La figura III.9, representa gráficamente el 100% de las iteraciones con los individuos más aptos de la generación 34.

Los puntos en cada mejor generación fueron variando para las generaciones intermedias (esto es, 19 para la generación 0 y 23 para le generación 2). Nunca se especifico, que la solución debería tener 20 puntos, ni la forma o el contenido particular de la expresión-S, ni tampoco, que aparecería en la generación 34. El tamaño, la forma y el contenido de la expresión-S que resuelve este problema fue determinada por la presión selectiva que ejerció la medida de aptitud establecida (error).

La función obtenida está completa en el sentido de que está definida por algún punto en el intervalo original [-1, +1] y puede verse como un modelo del proceso efectuado con los 20 puntos del conjunto de datos.(i.e., los 20 casos de aptitud).

El individuo más apto, empleó las funciones +, -, * y cos. No usó %, sin, EXP ni RLOG; es decir, 4 de las 8 funciones primitivas en el conjunto fueron extrañas.

Conclusiones

La Computación Evolutiva se abre como una serie de nuevas técnicas, en el universo de la Computación, robustas y eficaces para la optimización de recursos.

Estas técnicas (Algoritmos Genéticos, Sistemas Evolutivos, Programación Genética) todavía son abordadas por revistas especializadas, coloquios, seminarios y simposiums internacionales.

En nuestro país han tomado auge y el Laboratorio Nacional de Informática Avanzada (LANIA) así como diversos investigadores independientes (Fernando Galindo Soria[7], Edgar González[8], Santiago Negrete, Pablo Noriega, Carlos Zozoya Gorostiza[9], Fernando Estrada de la Concha[10], entre otros) han realizado trabajos muy interesantes en éste campo en los últimos 10 años.

Hoy, éste trabajo pretende crear el interés suficiente, para el estudio de estas técnicas, cuyo valor, como herramientas para mejorar resultados en muchas áreas donde se requiere precisión y tener la posibilidad de que la programación sea más eficaz y adaptable a los cambios que surjan en algún momento dentro de un ambiente específico; es considerablemente alto.

Como egresada de la Licenciatura en Matemáticas Aplicadas y Computación, esta investigación, es una invitación a los futuros alumnos de Temas Selectos y a todos aquellos

que deseen mirar un poco más del potencial de una máquina, un lenguaje de programación y de todo el valor de un modelo matemático. Donde la abstracción de una técnica matemática se convierte en una robusta técnica de optimización inspirada en los mecanismos de selección natural.

Involucrando diversos conceptos que muestran la infinidad de posibilidades de desarrollo de un egresado de Matemáticas Aplicadas y Computación.

[7] Lic. Fernando Galindo Soria. Lic. en Ciencias de la Informática. UPIICSA-IPN.

[8] Edgar González, Santiago Negrete y Pablo Noriega. Centro Científico de IBM, México.

[9] Carlos Zozoya Gorostiza. Doctor en Ingeniería Civil de la Universidad de Carnegie Mellon, E.U.A. Actualmente es el Director General de la División Académica de Computación del ITAM. Sus áreas de interés son la Inteligencia Artificial, la Optimización y el Impacto de la Tecnología de Información en las Organizaciones.

[10] Luis F. Estrada de la Concha. Ingeniero en Computación del ITAM. Maestro en Ingeniería Agrícola en la Universidad de Arizona, E.U.A.

Bibliografía

[1] Buckles, B.P. and Petry, F.E. "Genetic Algorithms", IEEE Computer Society Press, 1992.

[2] Estivil, V. "Manual de Algoritmos Genéticos", Reporte Técnico, IX Reunión de Inteligencia Artificial, Veracruz, Ver., México, 1992.

[3] Estrada de la Concha, L.F. "Uso de Algoritmos Genéticos para Optimizar el Costo de Arneses Eléctricos Automotrices", Tesis de la Licenciatura en Ingeniería en Computación, ITAM, División Académica de Computación, 1994.

[4] Galindo Soria, F. "Sistemas Evolutivos", Reporte Técnico, VI Reunión de Inteligencia Artificial, Querétaro, Qro., 1989.

[5] Galindo Soria, F. y García, M.A. "Programación Dirigida por Sintaxis", Reporte Técnico, V Reunión Nacional de Inteligencia Artificial, Mérida, Yucatán, Marzo, 1988.

[6] Glenn Brookshear, J. "Teoría de la Computación: Lenguajes Formales, Autómatas y Complejidad", Addison-Wesley Iberoamericana, México, D.F. , 1993.

[7] Goldberg, D.E. "Genetic Algorithms in Search Optimization and Machine Learning", Addison-Wesley, 1989.

[8] Guerra, Salcedo César. "Resolviendo Consultas de Apareamiento Parcial Utilizando Algoritmos Genéticos", Reporte Técnico, VII Reunión Nacional de Inteligencia Artificial, Ensenada, Baja California Sur, Septiembre, 1991.

[9] Holland J.H. "Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence", The MIT Press, 1992.

[10] Kaufmann, A. y Faure, R. "Invitación a la Investigación de Operaciones", Segunda Edición, CECSA, México, 1977.

[11] Koza, J.R. "Genetic Programming -On the Programming of Computers by Means of Natural Selection", The MIT Press, 1992.

[12] Pérez, Alvárez Mario. "Relajamiento de Agregados Atómicos Usando Algoritmos Genéticos", Tesis de la Licenciatura en Matemáticas Aplicadas y Computación, ENEP Acatlán, UNAM, 1996.

[13] Ribeiro Filho, J.L. "Genetic-Algorithm Programming Environments", IEEE Computer, June 1994.

[14] Sanchis Llorca, F.J. "Compiladores: Teoría y Construcción", Segunda Edición, PARANINFO, Madrid, España, 1988.

[15] Ullman, Jeffrey D. "Principles of Database and Knowledge-Base Systems", Computer Science Press 1988.

[16] Whitley, Darrell, Kauth Joan. "GENITOR: A Different Genetic Algorithm", Technical Report CS-88-101, Department of Computer Science, Colorado State University, 1988.