Algoritmos Genéticos

Esteban Di Tada *

Introducción

En el año1831 Charles Darwin navegó hasta la islas Galápagos, ubicadas aproximadamente a 960 Km al este de la costa de la República del Ecuador, a bordo del barco de bandera inglesa HMS Beagle. Su capitán era Robert Fitzroy, un descendiente ilegítimo del rey Carlos II.

Darwin sólo tenia 20 años cuando dejo Inglaterra en 1831. Treinta años más tarde publicaría su teoría de la evolución la que, indudablemente, fue una de las ideas más revolucionarias que la ciencia ha tenido *(On the Origin of Species by Means of Natural Selection* 1859). Aun en nuestros tiempos se discute su validez, no tanto por razones científicas, sino debido a principios, en general, de carácter religioso. No abordaremos en este trabajo las razones por las cuales se acepta o no la teoría de Darwin, sino que se desarrollará una serie de posibles aplicaciones a partir del principio de adaptación por él planteado.

Una de las preguntas que Darwin se formuló al observar la flora y fauna de las islas, fue como tantas especies de plantas y animales diversos habían llegado a las mismas.

Las corrientes marinas, con sus cambios periódicos, podrían haber permitido que muchas diferentes especies inmigraran a las islas. Algunas, como los leones de mar, las focas y los pingüinos, podrían haber nadado y llegado de este modo, ayudados, con toda seguridad, por las corrientes marinas. Otras, como las tortugas gigantes, podrían haber flotado en el mar y ser arrastrados hasta las islas por las corrientes marinas

Las esporas livianas de muchas plantas podrían haber llegado llevadas por el viento conjuntamente con algunos vegetales vasculares con semillas livianas. Arácnidos, pequeños insectos y pequeños caracoles terrestres son, frecuentemente, transportados por el viento.

Las aves terrestres y murciélagos, malos voladores, no hubieran podido llegar volando, sino que debieron, seguramente, llegar a las islas llevados por el viento. Las aves marinas podrían haber llegado fácilmente volando desde tierra.

Las aves muchas veces colaboran con las plantas transportando sus semillas en su estómago y expulsándolas en su destino. Semillas con pequeños ganchos pueden ser transportadas en las plumas de las aves. Otras semillas pueden viajar, mezcladas con barro, adherido a las patas o plumas de aves, pudiendo, de esa manera, haber llegado a las islas.

A Darwin le asombro la cantidad de diversas especies que había en las islas.

Fue entonces cuando desarrolló su teoría de la selección natural, por medio de la cual los seres más fuertes y adaptados al hábitat sobreviven, aumentando, de esta manera, la fortaleza general de la especie en relación con el entorno.

---

* Ingeniero Aeronáutico (Ecole Superieur de L´Aeronautique Française, París). Master of Science in Electrical Engineering (Universidad de Purdue, Indiana, USA). Miembro de la sociedad científica Sigma Xi. Decano de la Facultad de Ciencia y Tecnología de la Universidad de Palermo.

Llegadas las diferentes especies a las islas, se establecieron en ellas y determinaron sus territorios. La evolución comenzó y muchas especies únicas se generaron como, por ejemplo, los pinzones de Darwin.

Los pinzones, probablemente, descienden de algún ancestro común. Luego, debido al aislamiento en las islas, a fenómenos probabilísticos, a climas diferentes, a características naturales tales como disponibilidad y tipo de alimentos, se derivaron en trece diferentes especies.

El proceso de su evolución debe de haber comenzado con inmigrantes del continente. A medida que se dispersaron a diferentes islas nuevas poblaciones se formaron. Cada vez que las poblaciones satélites se dispersaban se profundizaban las diferencias entre las especies. Y sólo aquellos más aptos sobrevivían y podían transmitir a sus descendientes sus características.

Otro de los grandes de la historia que contribuyeron a explicar el fenómeno de la adaptación y evolución fue Gregor Mendel cuyas teorías sobre la herencia basadas en sus investigaciones con vegetales, son conocidas por cualquier estudiante de biología. Sin embargo su trabajo fue tan brillante y sin precedentes que en la época que lo expuso requirió mas 34 años para que el resto de la comunidad científica lo reconociera. Su breve monografía ***"Experimentos con plantas híbridas"*** se ha transformado en una perdurable e influenciadora publicación en la historia de la ciencia. Mendel, la primer persona que analizó las características de generaciones sucesivas de seres vivos, no fue un científico mundialmente reconocido. Fue un monje Agustiniano que enseñaba ciencias naturales a alumnos de colegios secundarios y su atracción por la investigación estaba basada en su profundo amor por la naturaleza.

No se puede dejar de mencionar, en este breve resumen, las contribuciones hechas por Pasteur, las hechas en 1953 por James Watson y Francis Crick que propusieron una estructura para la molécula de DNA la que, no solo explicaba el apareo de las bases, sino que ofrece un modelo relativamente simple de cómo se almacena y transfiere la información genética y toda una serie de importantes científicos que estudiaron el problema de la genética, abriendo la puerta para la cura de innumerables enfermedades.

El avance en el conocimiento del proceso de transmisión de información genética, conjuntamente con el incremento de las capacidades de las computadoras, hicieron posible aplicarlas tanto en la simulación problemas de biología como para la solución de problemas de optimización y búsqueda en diferentes disciplinas de la tecnología y la ciencia.

A mediados del siglo anterior (por el año 1960) diversos investigadores retomaron las ideas de Darwin para, por medio de las computadoras digitales, simular los procesos evolutivos. Podemos mencionar a I. Rechenberg en su trabajo ***"Estrategias de Evolución"*** (Evolutionsstrategie). Su idea fue luego desarrollada por otras investigadores. Los algoritmos genéticos ***(Genetic Algorithms: GA)*** fueron inventados por John Holland y desarrollados por él y sus estudiantes y colegas. Esto llevo a su hoy ya clásico libro Adaptación en Sistemas Naturales y Artificiales ***(Adaption in Natural and Artificial Systems)*** publicado en el año 1975.

En 1992 John Koza empleó los algoritmos genéticos para hacer evolucionar programas para realizar ciertas tareas. El bautizó este método como Programación Genética ***(Genetic***

Programming: GP). Los programas eran generados en el leguaje LISP, dado la facilidad de representar los mismos por medio de árboles sintácticos con los que los algoritmos definidos por Koza trabajan.

Principios Básicos

Los algoritmos genéticos pertenecen a la clase de métodos de búsqueda aleatoria. Su diferencia fundamental con los procedimientos clásicos es que los algoritmos genéticos se basan en la evolución de una familia de soluciones (generación) en lugar de ir mejorando una sola de las soluciones.

El principio fundamental de funcionamiento es el siguiente:

Cada solución se identifica por una cadena de bits, que será llamada cromosoma por analogía con el mecanismo de reproducción de seres vivos. La solución de un problema debe, por lo tanto, ser codificada como un string (cadena) de bits (o otra estructura de datos). Inicialmente se crea de manera aleatoria una familia de soluciones (cromosomas o genomas) que constituirán la generación inicial (Como la primer inmigración de pinzones que llegaron a las islas Galápagos). A partir de esta se crean sucesivas generaciones de soluciones de manera tal que las mismas se vayan "adaptando" a las condiciones impuestas. Estas condiciones pueden ser la optimización de una función que dependa de la solución o cualquier medida que defina, en cierta manera, la bondad de la solución. Como se desprende del párrafo anterior, el método no requiere hacer ninguna hipótesis sobre el modelo que rige el comportamiento de las soluciones.

Por lo tanto para aplicar esta técnica es necesario:

- Representar cada solución por medio de una cadena de objetos o algo similar.
  El mecanismo que se elija depende del tipo de problema a resolver. Una buena elección de la codificación de soluciones en cromosomas facilitará enormemente la tarea posterior.
- Definir el nivel de adaptación de cada solución
  La forma de evaluar el coeficiente de adaptación de las soluciones depende del tipo de problema a resolver. En general se trata de elegir una función que vaya de 0 a 1 (normalizada).
- Obtener una nueva generación a partir de la anterior.
  Este mecanismo de reproducción debe ser diseñado de manera tal que la próxima generación sea "mejor" que la anterior. El término mejor debe entenderse como que los individuos que la constituyan tengan un coeficiente de adaptación (menor costo, mayor beneficio, mejor control, mayor rapidez) superior a la anterior.

La solución al problema será aquella que tenga el mejor coeficiente de adaptación.

Los algoritmos genéticos no garantizan la obtención de un óptimo sino una solución "aceptable".

Una de las grandes ventajas de este método de optimización es su simplicidad. A pesar de su sencillez pueden aplicarse a una gran cantidad de problemas prácticos que aparecen a diario en la ciencia, la tecnología y la vida cotidiana.

Existe una serie de modificaciones sobre el mecanismo original que puede aplicarse para una mejora del proceso. Básicamente si la codificación y la función de adaptación están bien elegidas, las variaciones que se apliquen solo permiten obtener mejoras menores.

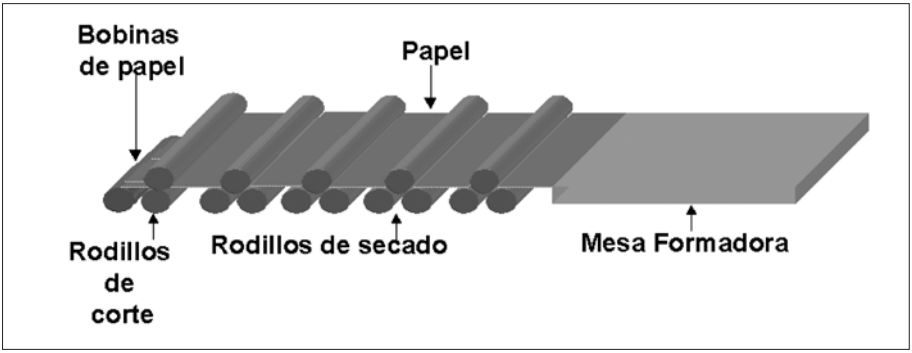
Dado que el algoritmo esta separado de la representación, la búsqueda de soluciones de problemas mixtos continuos/discretos es tan sencilla como la búsqueda de soluciones continuas o discretas.

Uno de los mayores inconvenientes es la gran cantidad de cálculo que requieren. Pero su propio principio de funcionamiento los hace, como las redes neuronales, especialmente adaptados para el procesamiento paralelo.

Descripción

Un Ejemplo

Para facilitar la descripción del funcionamiento de los Algoritmos Genéticos se empleará un ejemplo. Sea una planta de producción continua de papel (lo que se llama, en la jerga de dicha industria, una máquina de papel). La misma consiste de un mesa formadora en donde se pone la pasta de papel (Fibras celulósicas más otros componentes). La mesa formadora alimenta una serie de rodillos calentados a alta temperatura que secan y comprimen la pasta, generando, de esa manera, el papel. Se pueden producir diferentes colores y calidades de papel. Para cada par de estos corresponden capacidades máximas de producción. (El papel se mide por el gramaje que es el peso en gramos de un metro cuadrado del mismo). Cuanto más alto es el gramaje, menos es la producción horaria, debido a las limitaciones impuestas por la máxima capacidad de trabajo de la máquina. Cuanto más gramaje mayor es el flujo de masa y por lo tanto mayor es la necesidad de potencia para generar el suficiente calor para secarlo y la suficiente presión para laminarlo y llevarlo al espesor deseado. A menores gramajes (papel más fino) la limitación esta dada por la resistencia a la rotura del papel, ya que el aumento de la capacidad de producción requeriría velocidades que harían que el papel se rompiera obligando a parar la planta y reiniciar el proceso de producción. En el gráfico siguiente se presente un esquema de la organización de una máquina de papel.



Cuando se cambia el color del papel a producir, es necesario limpiar la máquina. Esto tiene un costo que está compuesto por:

- Los costos directos (personal y equipos afectados al proceso)
- Los costos indirectos de lucro cesante de la amortización de la máquina.
- El costo de limpieza de la máquina. Si se produce papel negro y luego blanco, por ejemplo, el costo será mucho más elevado que si se produce primero blanco y después negro (La limpieza de la máquina para pasar del negro al blanco demora mucho mas que del negro al blanco por lo que el costo de cambio de producción es mayor).

Finalmente existen costos de orden financiero y de comercialización. Los clientes formulan pedidos los que se definen por medio de una cantidad, gramaje y color de papel a lo largo del tiempo. Si el papel se le entrega en el término por ellos requerido, se obtiene un ingreso y un cliente satisfecho. Si no se entrega no se cobra y el cliente esta descontento, lo que implica un costo de comercialización importante, máxime si se trata de commodities que pueden ser reemplazadas en el mercado interno o externo. Por otro lado, si se produce el papel con antelación para formar stock, se incurrirá en un costo financiero por la inmovilización de capital que ello involucra.

El siguiente cuadro explica la estructura de los costos:

	Se produce de más	***Se produce la cantidad exacta***	***Se produce de menos***
Costos financieros	Grandes (por producción inmovilizada y/o inutilización del producto si es perecedero)	Mínimos	Mínimos
Costos comerciales	Nulos	Nulos	Mayores (por clientes insatisfechos)
Costos de producción	Mayores (sobre todo si el producto es perecedero)	Estándar	Menores
Ingresos por ventas	Iguales	Iguales	Menores

A estos costos se le deben sumar los costos de limpieza de la máquina, que, como se ha dicho anteriormente, no son conmutativos (dependen del orden de producción de colores y, en general de los atributos que definen la calidad).

El problema que se plantea es el de obtener un plan de producción que minimice los costos, o mejor dicho, optimice los ingresos.

El problema es complicado de resolver por los métodos tradicionales. Sin embargo su asimilación a un problema adaptado al empleo de algoritmos genéticos es relativamente fácil. En efecto, supóngase que se deban planificar 50 días de producción y que no sea conveniente cambiar el tipo de papel a producir en un día. (La experiencia indica que, en base al tiempo mínimo que requiere la limpieza, no es conveniente cambiar de tipo de papel dentro del día). Cada solución se puede representar por medio de un vector de 50

entradas. A cada una de las entradas se le asignará un código correspondiente al color del papel (solo el color se tendrá en cuenta en este ejemplo) que se desea producir y la cantidad requerida por los clientes (Para simplificar el ejemplo no se ha incluido en el mismo el gramaje del papel, pero su incorporación no modifica las conclusiones que se obtengan).

La siguiente tabla muestra un ejemplo de la representación de un requerimiento de producción:

Día	*Código de Color*	*Cantidad pedida en toneladas*
1	0	10
2	0	23
3	0	12
4	1	5
5	1	23

Ella corresponderá (Si el código de blanco es 0 y el de negro es 1) a una producción de papel blanco los tres primeros días (10, 23 y 12 toneladas requeridas), seguida de dos días de producción de papel negro (5 y 23 toneladas requeridas).

Si el periodo de un día fuera demasiado grande, se podrían emplear periodos de planificación más cortos o, si fuera a la inversa, más grandes.

Generación de la población inicial

En este caso la población inicial es muy sencilla de obtener. Las asignaciones del color a producir y la cantidad para cada día se generarán al azar teniendo en cuenta la cantidad total a producir de cada color y la capacidad máxima de producción de la máquina. Es de hacer notar que la incorporaciones de limitaciones de producción por color y/o gramaje serán sencillas de introducir.

Generación de las poblaciones sucesivas

Definido el mecanismo de codificación del problema, es necesario establecer el procedimiento a emplear para la obtención de la próxima generación a partir de la corriente. Para ello se copia a la naturaleza. Dos son los mecanismos básicos que existen:

- La reproducción y
- La mutación

En el primero, por medio de uno o dos integrantes de la generación actual, se crean uno o dos nuevos integrantes de la próxima generación. Una vez finalizado este proceso de reproducción se destruye la anterior generación la que es sustituida por la recientemente creada. Dos son las preguntas básicas que se presentan:

- Como elegir el o los padres o madres y
- Como crear los hijos

La respuesta a la primer pregunta es sencilla. La elección se realiza con una probabilidad que es una función creciente del coeficiente de adaptación de cada individuo (Una función es creciente si un aumento del argumento genera un aumento en el valor de la función). De esta manera las soluciones más adaptadas serán elegidas más frecuentemente. La elección de los padres es con repetición, es decir un mismo individuo puede ser seleccionado varias veces en el proceso de un período. La forma más sencilla es emplear la función identidad, es decir seleccionar los padres en forma proporcional al coeficiente de adaptación. Si se empleara la función logaritmo se suavizaran las grandes diferencias. Por ejemplo en la tabla siguiente se muestra las diferencias que existen entre el uso directo del coeficiente, el uso de su cuadrado y el uso del logaritmo (El cuadrado exacerba las diferencias en tanto que el logaritmo las atenúa)

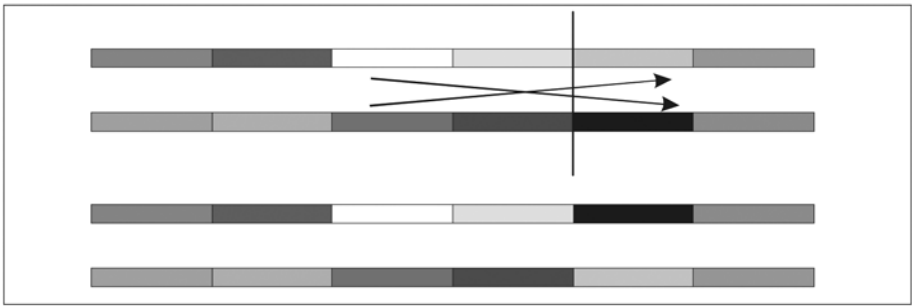
Coeficiente de adaptación		*Cuadrado*		*Logaritmo*	
Valor	***Diferencia***	***Valor***	***Diferencia***	***Valor***	***Diferencia***
4		16		2,77	
12	8	144	128	4,96	2,19

Se puede notar que la diferencia de 8 en el caso de emplear el coeficiente de adaptación directamente, se aumenta a 128 en el caso del cuadrado y se reduce a 2,19 en el caso del logaritmo.

En el ejemplo dado cada solución (individuo) tiene un solo cromosoma (haploide) en oposición al caso de otros seres que tienen varios pares de cromosomas (diploides).

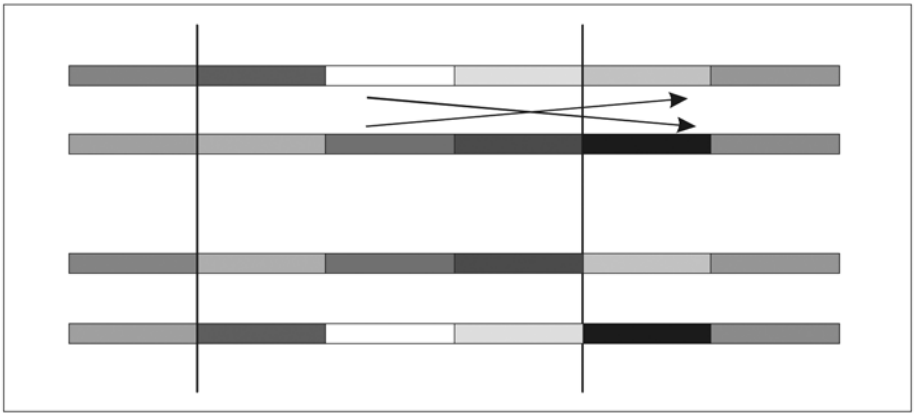
La reproducción no sexuada consiste en transferir el padre a la nueva generación. La reproducción sexuada (Si bien las soluciones no tienen sexo, es decir que no se dividen en dos particiones de manera tal que la reproducción se obtiene eligiendo un individuo de cada una de dichas particiones) se realiza por medio del mecanismo de sobrecruzamiento (crossover). Hay varios métodos. Él más sencillo es dividir ambos cromosomas en un punto (eligiendo un número al azar menor que la longitud de cada uno). Se forman así dos nuevos cromosomas (dos nuevos individuos) de la siguiente manera:

El primero con la primer parte del cromosoma del primer ascendiente (que arbitrariamente se llamara padre) colocando a continuación la segunda parte del cromosoma del segundo ascendiente (que arbitrariamente se llamara madre). Así se forma el primer hijo. El segundo hijo se forma con la primer parte de la madre y a continuación la segunda parte del padre.



El dibujo anterior ejemplifica este mecanismo.

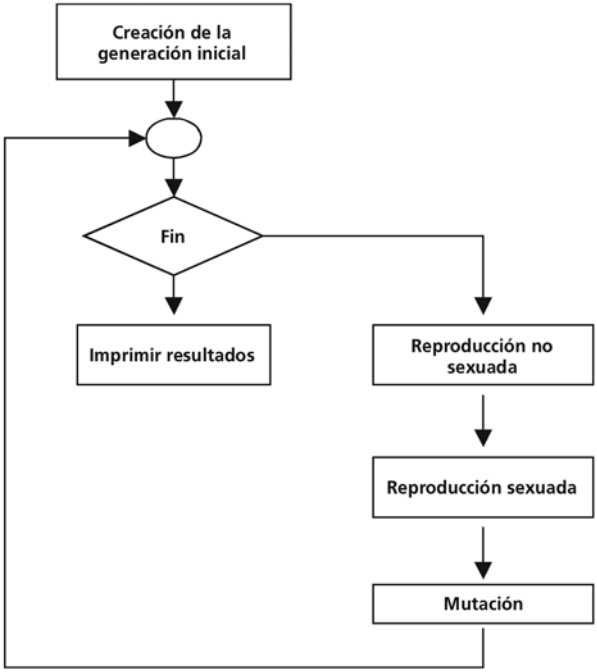
Métodos más complejos pueden ser empleados. Uno de ellos consiste en dividir al azar en tres segmentos, en lugar de dos como en el método anterior, los cromosomas de padre y madre, e ir asignándolos alternativamente a cada uno de los dos hijos. En el gráfico adjunto se ejemplifica este método.



El método descripto anteriormente puede extenderse fácilmente y dividir al cromosoma en n segmentos. Los hijos se obtendrán seleccionando dichos segmentos en forma alternativa entre el padre y la madre.

El otro operador que actúa es el de la mutación. Es bien sabido por los criadores de ganado, que es necesario mantener la diversidad genética dentro de un grupo para garantizar una mejora de la especie. Si no se toma esa precaución las generaciones se homogeneizan terminado todos los individuos iguales. Al llegar a dicha situación es imposible seguir mejorando el coeficiente de adaptación, dado que si todos los cromosomas son iguales y la reproducción se realiza por medio de los métodos descriptos anteriormente, se obtendrían todos los descendientes también iguales. La mutación, que es el proceso por medio del cual se realiza un cambio aleatorio de un individuo tiende a resolver este problema. Este operador, elige al azar un gen de un individuo (Con muy baja probabilidad del orden del 0.001) y lo altera al azar. En el caso real es conveniente que se mida la homogeneidad de la población. Esto puede hacerse mediante la varianza del coeficiente de adaptación que solamente es cero cuando todos los coeficientes sean iguales. En general es conveniente que la probabilidad de mutación sea función de la varianza. Si la misma es muy baja se incrementa la probabilidad y si fuera muy alta se la disminuye, de manera tal de mantenerla en valores razonables.

En el diagrama de flujo adjunto se muestra la estructura básica del algoritmo.

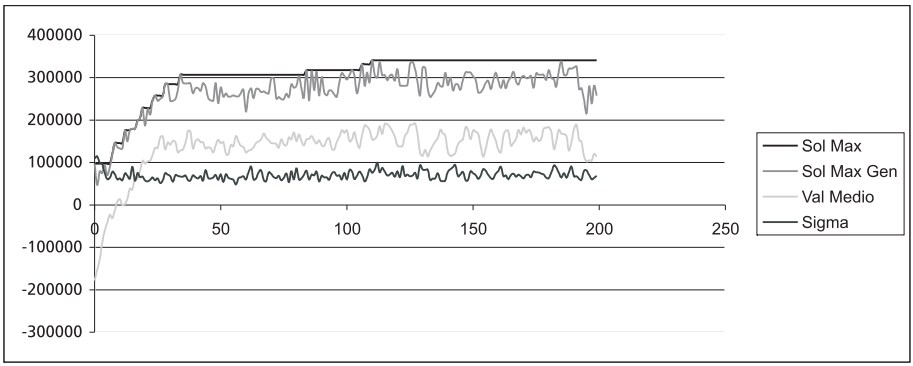


Siguiendo el esquema descrito en el diagrama anterior se construyó un programa en C++ y se obtuvieron los siguientes resultados para los siguientes valores de los diferentes parámetros:

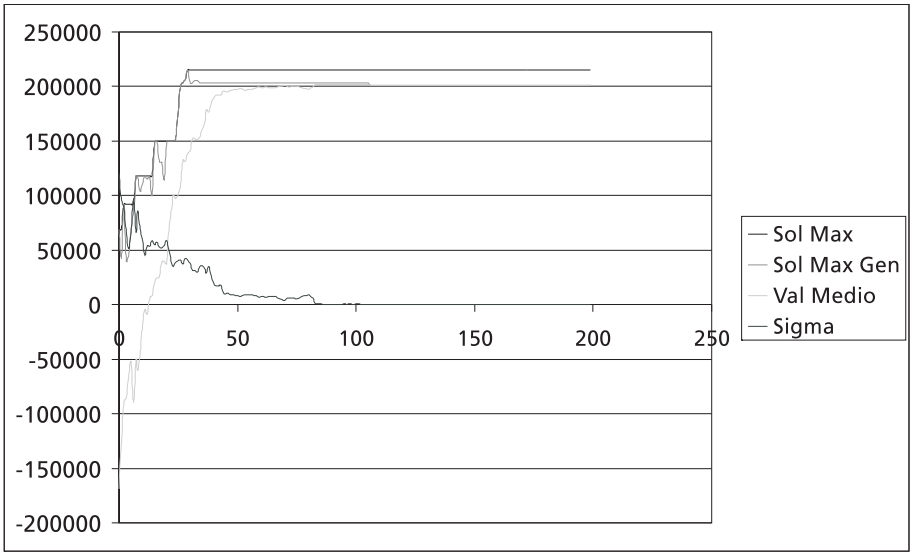
Dimensión de la población	200
Cantidad de períodos	50
Cantidad de colores	3
Probabilidad de mutación	0.01
Cantidad de generaciones	100
Probabilidad Inferior de mutación	0.01
Probabilidad superior de mutación	0.2
Costo de ruptura de stock	300
Precio Color 0	500
Precio Color 1	200
Precio Color 2	100
Costo color 0	150
Costo color 1	90
Costo color 2	60
Costo cambio de color de 0 a 1	100
Costo cambio de color de 0 a 2	200
Costo cambio de color de 1 a 0	10
Costo cambio de color de 1 a 2	100
Costo cambio de color de 2 a 0	20
Costo cambio de color de 2 a 1	40

En los gráficos siguientes se muestra la evolución si el control de probabilidad de mutación se elimina. Con control de mutación se obtiene un máximo de aproximadamente 350000 que se logra en la generación 140. Si dicho control se elimina en la generación 80 la varianza llega a cero y el máximo que se obtiene es de solamente 210000.

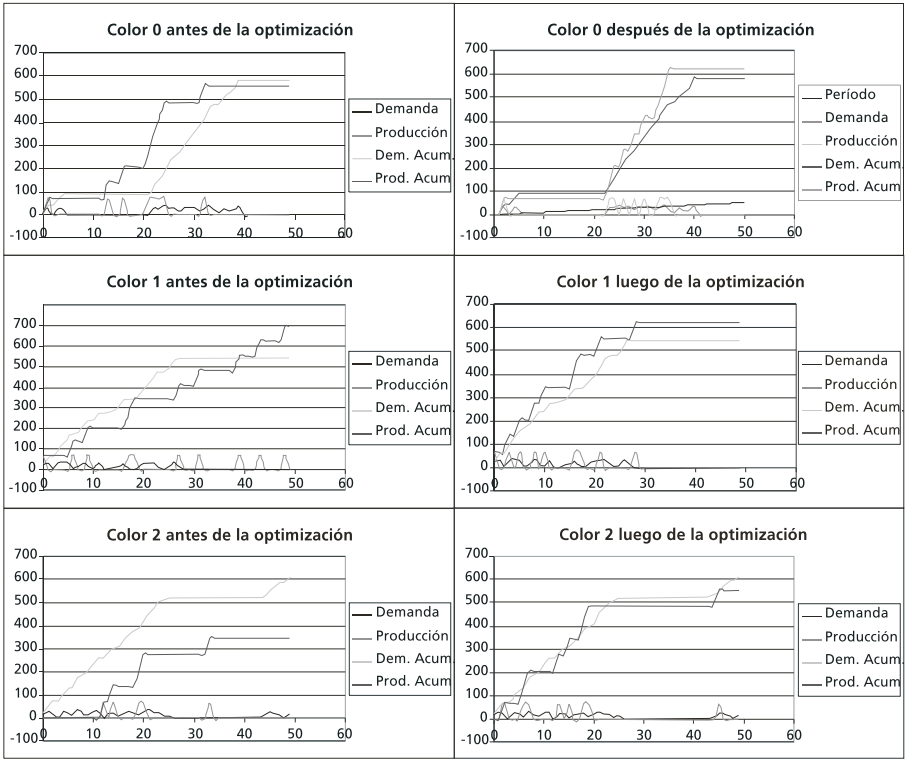
Con ajuste de la probabilidad de Mutación



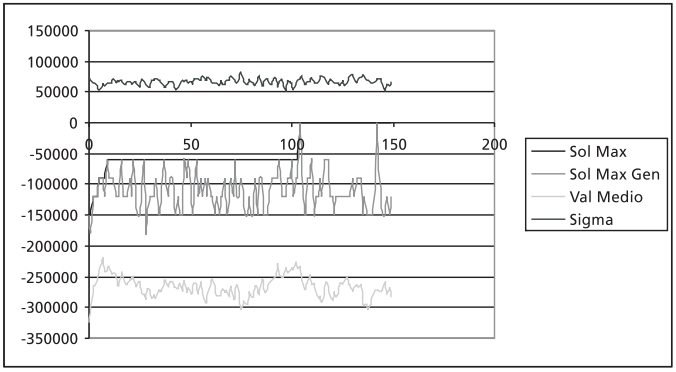
Sin ajuste de la probabilidad de mutacion

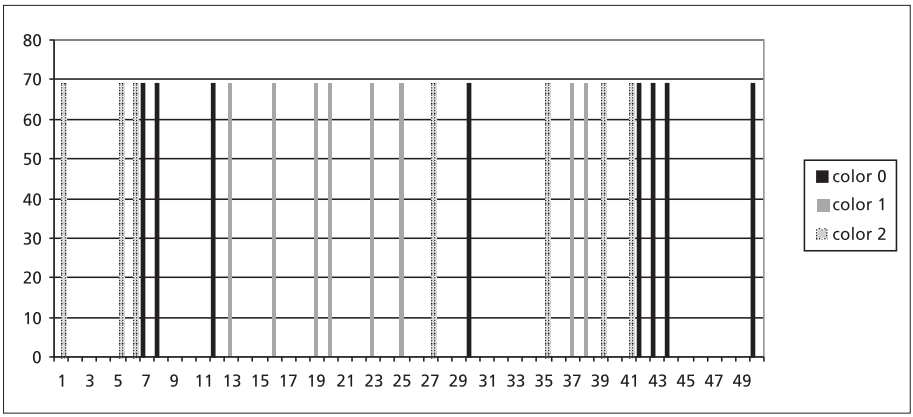


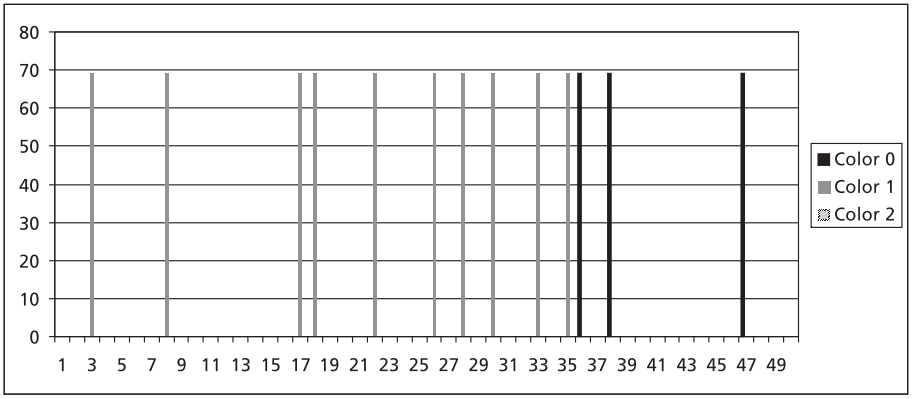
En los siguientes gráficos es posible observar como las curvas de producción se van adaptando a los requerimientos para los diferentes colores



Finalmente es posible observar como se reordenan las producciones para evitar los costos por limpieza de la maquina. Los datos de costo de limpieza fueron exagerados de ex profeso en el ejemplo para que perdieran importancia en relación con el resto de los costos de limpieza de la máquina y mostrar la capacidad de ordenamiento del algoritmo.







Programación Genética

Autómatas finitos

En los párrafos anteriores se expuso la manera por medio de la cual, utilizando los principios de transmisión hereditaria de caracteres, se puede optimizar la solución de ciertos tipos de problemas.

En realidad los algoritmos genéticos no permiten obtener resultados óptimos, sino que se obtienen soluciones aceptables, en el sentido que los valores de las funciones de adaptación adopten valores adecuados para las soluciones a los problemas planteados.

Se abordará ahora un problema más general que será llamado Programación Genética. Para ello se extenderá el mecanismo descripto anteriormente a la solución de problemas en los que el espacio de soluciones no son los parámetros que definen una solución, sino el espacio de algoritmos capaces de resolver un cierto problema. Conceptualmente el mecanismo de solución es el mismo. Se eligen algoritmos al azar con los que se

constituye la primer generación. Luego por medio de los operadores de reproducción y mutación se obtienen las sucesivas generaciones. La mejor de todas las soluciones obtenidas, será el algoritmo solución.

Se plantea entonces el problema de cómo representar algoritmos y como definir los operadores mencionados y de que manera se puede definir un coeficiente de adaptación de un algoritmo.

Para el primero de los problemas pueden existir dos soluciones que difieren en la manera por medio de la cual se definan los algoritmos. La primera consiste en definir como la estructura del algoritmo la de un autómata finito. Esto presenta dos problemas que son:

- Cuántos estados se asigna al autómata y
- Cómo se puede establecer un mapping entre dichos autómatas y un string de bits que represente un cromosoma.

El primero solo lo indica la experiencia o el ensayo de autómatas con diferentes cantidades de estados hasta obtener una solución aceptable. En cuanto al segundo problema, se planteará un esquema de mapping que garantiza que el crossover y la mutaciones generan nuevos strings que pueden convertirse en autómatas finitos.

La idea se basa en el trabajo de Jefferson y otros 1992[1] de acuerdo a como lo presenta Koza[2]. Se trata de encontrar un algoritmo que programe una hormiga artificial para que acceda a la comida que se encuentra distribuida en un cuadrado toroidal (tablero) de 32 x 32 divisiones.

El sendero de Santa Fe, como se lo conoce, es una sucesión de montones de comida distribuida siguiendo un camino con curvas. Cada unidad de comida (en total tiene 89) esta ubicada en uno de los cuadrados del tablero. El sendero no es continuo sino que tiene orificios (gaps) en los que no hay comida. Estos orificios (gaps) pueden ser:

- Simples
- Dobles
- Simples en las esquinas
- Dobles en las esquinas
- Triples en las esquinas

En el gráfico se representa el sendero de Santa Fe (en alusión al ferrocarril del Oeste de Estados Unidas).

Con color oscuro se indican los cuadrados en los que hay comida y con color claro los lugares en que no hay y que constituyen las discontinuidades antes mencionadas. La hormiga comienza en el cuadrado 1,1 mirando hacia la derecha.

La hormiga artificial puede ejecutar alguna de las siguientes acciones:

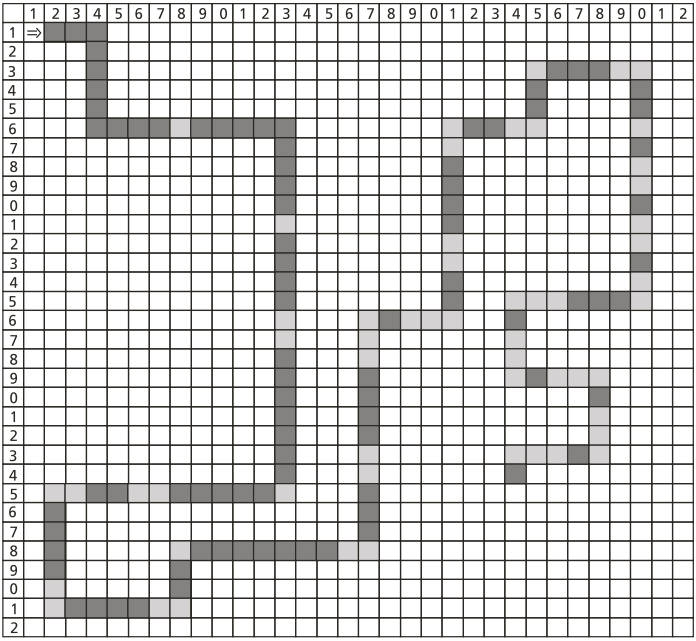
- Girar a la derecha 90º.
- Girar a la izquierda 90º.
- Avanzar en la dirección en que se encuentra. Si hubiera comida en la casilla a la cual avanza se la come y la elimina de la casilla. (la casilla queda sin comida).

---

1. Evolution as a theme in artificial life: The genesys/Tracker system. Langton, Christopher y otros, Artificial Life II. Addison-Wesley

2. Genetic Programming, John R. Koza. A Bradford Book, The MIT Press, Cambridge, Massachusetts pag. 54.

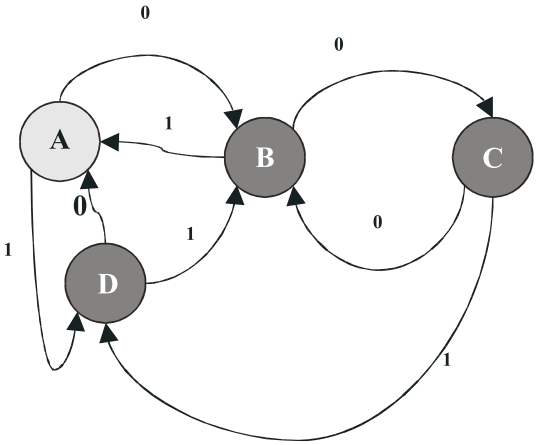
Su visión del mundo exterior es muy limitada. Solamente tiene un sensor por medio del cual puede decidir si hay o no comida en la celda inmediatamente adyacente a aquella en que se encuentra y sólo en la dirección en la cual esta avanzando.



El objetivo de la hormiga es recorrer todo el sendero (comiéndose toda la comida) en un tiempo razonable. Jefferson y Collings resolvieron el problema usando un autómata finito y una función biyectiva entre autómatas y un espacio de strings.

Sea el siguiente autómata representado por el grafo:

Estado	***Entrada*** ***0 No Hay Comida*** ***1 Hay Comida***	***Nuevo Estado***	***Operación***
A=00	0	B=01	01 (Derecha)
A=00	1	D=11	10 (lzquierda)
B=01	0	C=10	11 (Avanzar)
B=01	1	A=00	00 (nada)
C-10	0	B=01	11 (Avanzar)
C=0	1	D=11	10(izquierda)
D=11	0	A=00	11(Avanzar)
D=11	1	B=01	01 (Derecha)



El estado inicial es 00. El autómata anterior puede representarse por medio del siguiente string:

Acción			*		*		*		*		*		*		*		*
Estado Final		*		*		*		*		*		*		*		*	
Entrada		0		1		0		1		0		1		0		1	
Estado		00				01				10				11			
Estado Inicial	00	01	01	11	10	10	11	00	00	01	11	11	10	00	11	01	01
Posición	00	00	00	00	00	11	11	11	11	11	22	22	22	22	22	33	33
	01	23	45	67	89	01	23	45	67	89	01	23	45	67	89	01	23

Como se ve, un autómata de cuatro estados y dos entradas puede representarse por un string de 34 bits de longitud. La posición dentro del string del estado al que hace la transición y la acción están dadas por las siguientes fórmulas

$$Estado_{nuevo} = 2 + 8 * Estado_{anterior} + 4 * Entrada$$

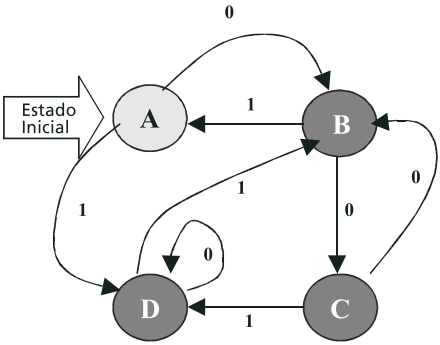
$$Acción = 4 + 8 * Estado_{anterior} + 4 * Entrada$$

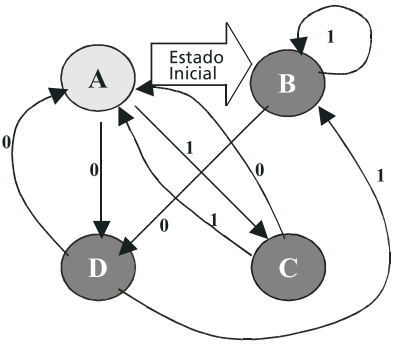
Se puede ver que a los dos siguientes autómatas

00	01	01	11	10	10	11	00	00	01	11	11	10	00	11	01	01
01	11	00	10	01	11	00	01	10	01	11	01	10	11	00	10	11

Les corresponde, por medio del operador de crossover, los siguientes dos autómatas:

00	01	01	11	10	10	11	00	00	01	11	11	10	11	00	10	11
01	11	00	10	01	11	00	01	10	01	11	01	10	00	11	01	01





La experiencia de Jefferson y Collins fue realizada con un autómata de 32 estados. Para representarlo se requiere un string de 453 bits. Obtuvieron una solución que permitió acceder a los 89 trozos de comida en 200 pasos.

Otro enfoque para representar algoritmos

Como se enunció en el párrafo anterior, una de las grandes limitaciones, que para representar algoritmos por medio de autómatas finitos existe, es la capacidad de proceso que este tipo de algoritmo tiene. En particular, es bien sabido que la capacidad de proceso de los autómatas finitos esta limitada al reconocimiento de expresiones regulares. No se puede construir un autómata finito que reconozca palíndromos de longitud arbitraria. Si se lo puede hacer para reconocer los que tienen una longitud que no puede superar un valor máximo prefijado de antemano. Pero aunque esta limitación no parezca demasiado grave, la estructura de un autómata que reconociera palíndromos de una longitud menor a 100 por ejemplo, seria excesivamente compleja. Por otro lado la estructura de un autómata stack que reconociera palíndromos seria muy simple.

Una solución es la de representar los algoritmos por medio de programas de tipo Lisp. Pero, que ventaja acarrea esta representación? La respuesta es directa. El Lisp es un lenguaje funcional y por ello resulta sencillo implementar los operadores de crossover y mutación.

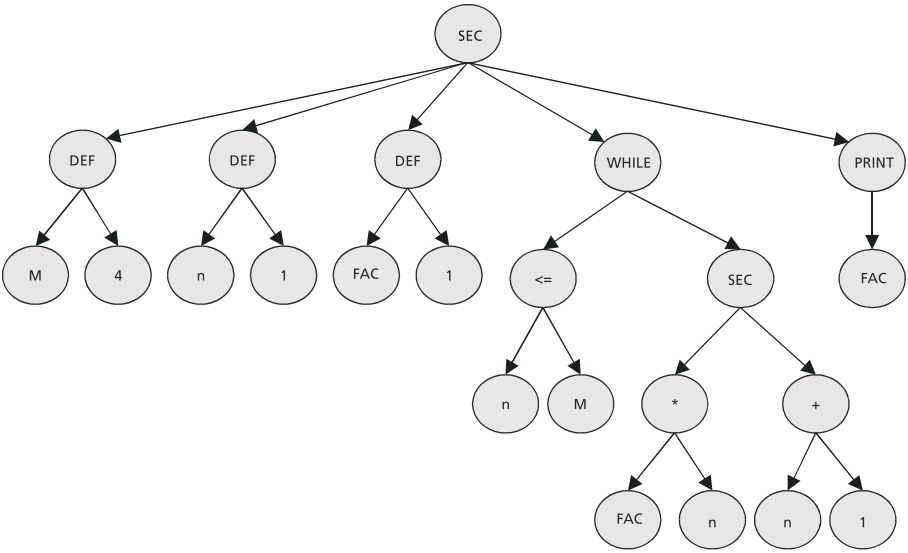
En efecto, un Programa Lisp es una función cuyos argumentos pueden a su vez ser funciones. Esta anidación de funciones puede representarse por medio de un árbol. El operador de crossover es simple de implementar. En cada uno de los padres se elige al azar un nodo. Luego los mismos se intercambian. Esto da la garantía que si ambos

padres eran formulas sintácticamente correctas, los hijos también lo serán. De la misma manera la mutación se realizará por medio de la elección al azar de un nodo el que se remplazara por un nuevo árbol (función) elegido al azar.

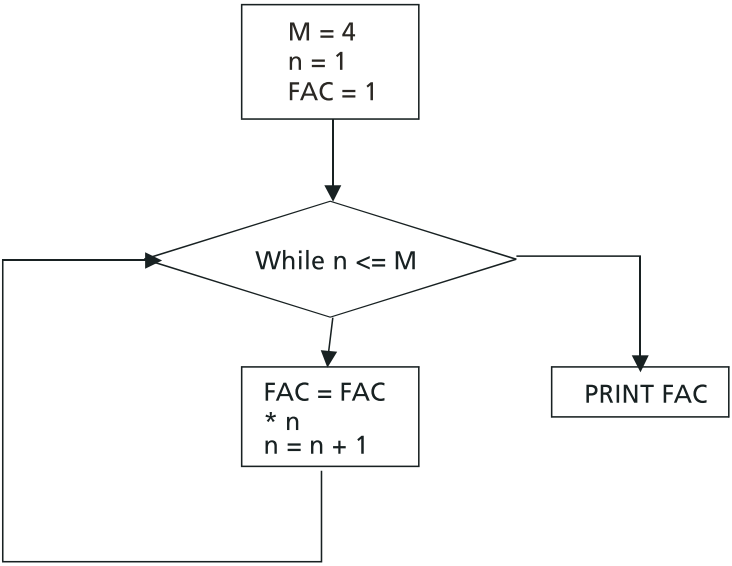
Considérese el siguiente ejemplo en el que el conjunto de funciones es el que se indica:

DEF Define una variable y le asigna un valor inicial Ej. (DEF X 2)
WHILE (while (<= A 2) (* X X)) Ejecuta (* X X)) mientras (<= A 2)
<= (<= A B) retorna verdad si a <= B y falso si no lo es
* (* A B) multiplica A por B y almacena el resultado en A
+ (+ A B) suma A mas B y almacena el resultado en A
PRINT (PRINT A) Imprime el valor de A
SEC Define una secuencia de funciones a ejecutar. Ej
(SEC (DEF A 1) (* A 2) (PRINT A))

Si se deseara escribir el factorial de n, la función seria:
(SEC (DEF M 4) (DEF n 1) (DEF FAC 1) (WHILE (<= n M) (SEC (* FAC n) (+ n 1))) (PRINT FAC)
El árbol que representa esta función es:



Y su representación como diagrama de flujo:



En el caso de la programación de la hormiga será necesario definir las siguientes funciones (que también se llamarán no terminales porque requieren la definición de los argumentos que usan):

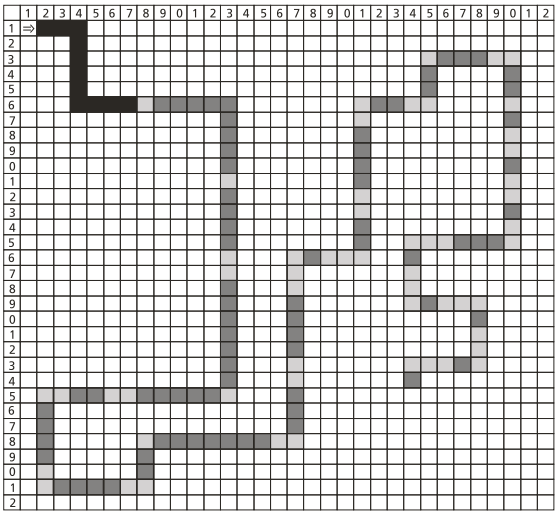
IF	Función con dos argumentos. Si en el cuadro adyacente en la dirección en que se encuentra la hormiga hay comida se ejecuta el primer argumento. En caso contrario, que no haya comida, se ejecuta el segundo argumento.
SEC	Función con n argumentos aj para j = 1,n. Se ejecutan en secuencia los n argumentos: a_1, a_2, ..., a_n

Y las siguientes acciones (que también se llaman terminales porque no requieren la definición de argumentos):

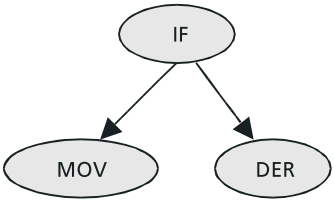
IZQ	Girar 90 grados a la izquierda
DER	Girar 90 grados a la derecha
MOV	Avanzar una casilla en la dirección en que se encuentra. Si hay comida comerla. La casilla de la cual se comió la comida se considerara vacía para sucesivas ejecuciones de la función IF.

Por ejemplo el siguiente programa

(SEC (IF (MOV) (DER))) generaría, suponiendo que la posición inicial es en la celda 1,1 y esta orientada hacia la derecha, el siguiente camino:

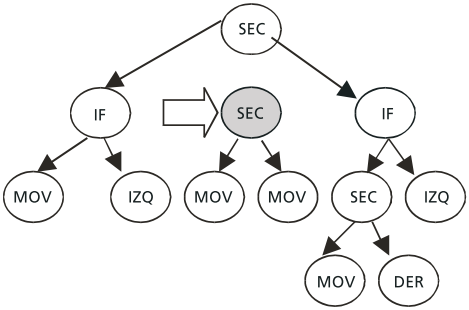


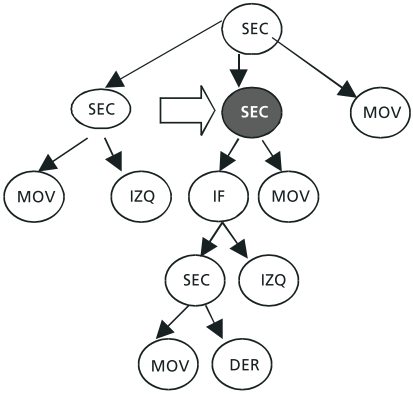
El camino seguido es el que esta pintado de negro. Como se puede apreciar una vez que llega a la casilla 6,7 comienza a girar en un ciclo infinito, dado que no existe comida en ninguna de las celdas adyacentes. El siguiente es el árbol que representa la fórmula



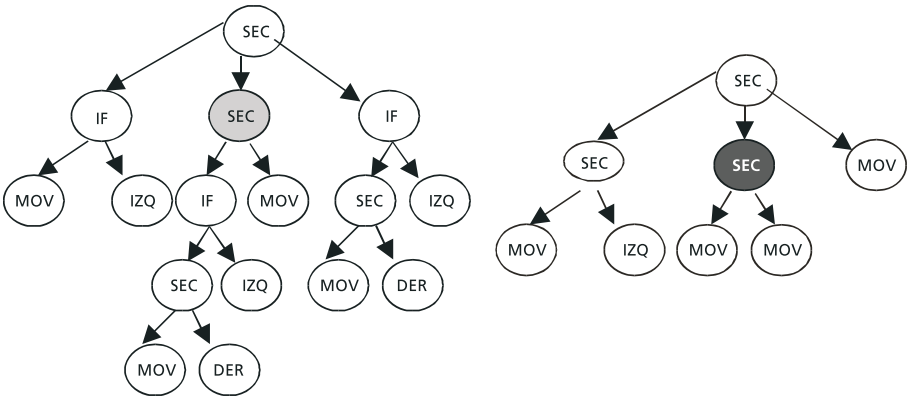
Operadores de crossover y mutación

Es interesante ver como funcionarían los operadores de crossover y mutación. Sean los siguientes dos "padres"





Si se eligen al azar los nodos marcados en cada uno de los padres y se intercambian los mismos (operador de crossover) los nuevos algoritmos serán los indicados a continuación:



En forma similar se puede definir el operador de mutación. Se elige al azar un nodo él que se reemplaza por una fórmula construida también al azar.

De esta manera queda resuelto el problema del crossover y la mutación, que son los dos operadores fundamentales del proceso de creación de una generación a partir de la anterior. Queda por definir el método a emplear para la creación aleatoria de fórmulas. Este método será empleado tanto para generación inicial como para las mutaciones sucesivas.

Creación de la generación inicial

La creación de la generación inicial es un problema desde el punto de vista teórico relativamente sencillo. Existen dos conjuntos de nodos[3]:

- Los nodos no terminales, como son en el caso en análisis las dos funciones (IF y SEC). Estos nodos contienen argumentos y por lo tanto no pueden ser hojas del árbol. Tienen que tener descendientes.
- Los nodos terminales, como MOV, DER, IZQ, que no contienen argumentos y que, por lo tanto, constituirán hojas del árbol (No tienen descendientes)

La primera idea sería hacer una lista de todos los elementos (Fórmulas y terminales) e ir eligiendo al azar elementos de la misma (Por ejemplo se ponen en un vector y se elige un número al azar entre 1 y la dimensión del vector). Cada vez que se elige una Fórmula, de manera recursiva, se eligen también a al azar las fórmulas que constituyen sus argumentos.

3. Para aquellos familiarizados con la teoría de lenguajes, estos nodos corresponden con los símbolos No terminales y terminales de la definición de una gramática $G = < N, \Sigma, S, P >$ donde N es el conjunto de no terminales, Σ el conjunto de terminales, S el símbolo inicial y P el conjunto de producciones.

Este método presenta las siguientes incógnitas:

- La probabilidad de elección debe ser la misma para todos los elementos o debe depender del elemento en cuestión (Por ejemplo debe haber más IF que secuencias)
- Si se elige como primer elemento una terminal tendríamos una fórmula muy elemental que, con toda seguridad, daría una solución mala (Por ejemplo si se eligiera DER la hormiga entraría en un lazo infinito de giros a la derecha)
- Si se llama profundidad de un nodo a la cantidad de arcos que es necesario recorrer para llegar a la raíz y profundidad de la fórmula a la máxima profundidad de los nodos que la constituyen, no esta garantizado que el método de creación al azar enunciado, genere fórmulas finitas.

Para el primer punto no existe, en nuestro conocimiento, una respuesta. Una solución sería crear al azar una distribución de probabilidades de elección de los elementos. Esto se lograría generando números al azar entre cero y 1 (flotantes) y luego normalizándolos.

La segunda y tercer punto están relacionadas entre si. Para ello definamos dos tipos de fórmulas[4], que son:

- Una fórmula completa de profundidad N como aquella en la que todos sus nodos terminales tienen profundidad N
- Una fórmula incompleta de profundidad N a aquella en la que uno de los terminales tiene profundidad N pero todo el resto tiene una profundidad menor que N

La solución adoptada fue la siguiente:

- Existe una profundidad de formula mínima (Que se eligió igual a 4). Esto impide la creación de fórmulas inútiles
- Se limita la longitud máxima de las fórmulas (Se empleo 17 siguiendo el ejemplo de Koza)
- Se generó con una probabilidad del 50% fórmulas completas e incompletas

Esto se logro cambiando el vector de probabilidades de elección de elementos.[5]

Función de adaptación

Existen varios criterios para la elección de la función de adaptación (fitness en inglés). En este caso su elección es sencilla. En efecto, la máxima cantidad de comida que una hormiga puede obtener es 89 unidades (De acuerdo con la definición del camino elegida). La función de inadaptación (que será mayor cuanto más "mala" sea la fórmula) será 89 menos la cantidad de comida obtenida. Será cero, por lo tanto, si come todo y 89 si no come nada. Si llamamos Fa a dicho valor, la función que se empleo en el ejemplo es $\Phi(f) = 1/(1+Fa)$ en donde f es la fórmula y Fa es 89 menos la cantidad de comida alcanzada.

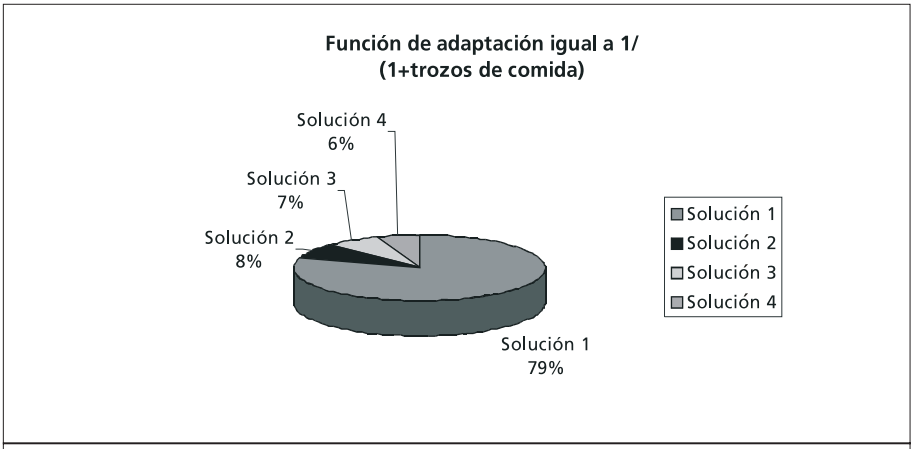
---

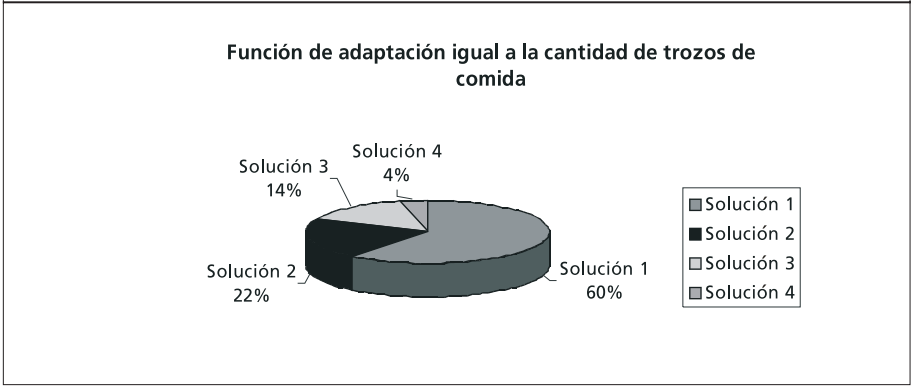
1. Ver Koza páginas 92 y 93
2. Por ejemplo para garantizar una profundidad mínina al comienzo la probabilidad de elección de terminales se la puso en cero, con lo que solo se pueden elegir nodos terminales. Para garantizar una profundidad máxima se fuerza, antes de llegar a la profundidad máxima, la probabilidad de elección de los nodos no terminales en cero. Dentro de cada tipo de elemento (Terminales y no terminales) los elementos eran equiprobables.

Para concretar las ideas precedentes se expondrá un ejemplo. Supóngase que existen cuatro soluciones cada una de las cuales logra obtener los trozos de comida que se indican en la tabla siguiente:

Solución	*Trozos de comida*	*Fa = 89 - trozos de comida*	$\Phi(f) = 1/(1+Fa)$
1	84	5	0,1667
2	30	59	0,0167
3	20	69	0,0143
4	5	84	0,0118

La forma de selección de una solución es con una probabilidad proporcional al valor de la función de adaptación. Un mecanismo físico sería el de construir una ruleta en la que cada uno de los sectores fuera proporcional al coeficiente de adaptación. Los gráficos siguientes muestran dicha ruleta:





La probabilidad de selección será:

Solución	*Trozos de comida*	$\Phi(f) = 1/(1+Fa)$
1	60%	79%
2	22%	8%
3	14%	7%
4	4%	6%

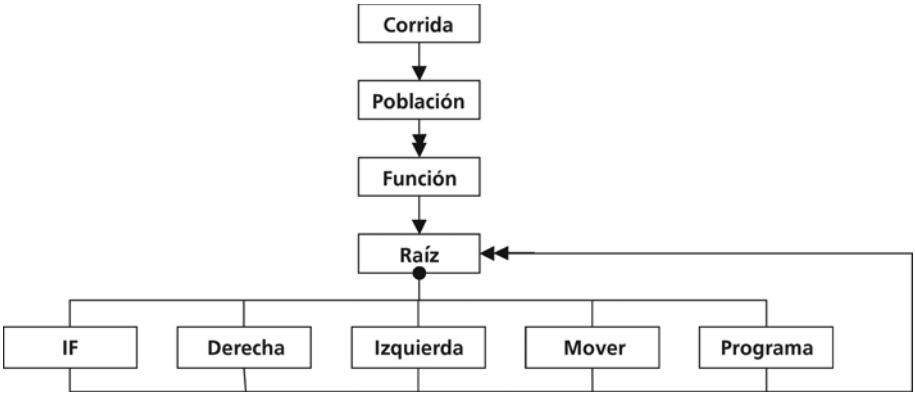
Como se ve en la tabla anterior, el uso de la función $\Phi(f) = 1/(1+Fa)$ hace que las buenas soluciones aparezcan como padres más frecuentemente (la primera pasa del 60% al 79%) en tanto que las intermedias decrementan su porcentaje de aparición y las malas o aumentan o quedan estacionarias.

En la simulación en computadora la elección se hace normalizando a 1 la suma de los valores de las funciones de adaptación y seleccionando cuando la sumatoria de las funciones de adaptación supere un valor elegido al random entre 0 y 1.

Implementación

El modelo se programó en C++ empleando el C++ Builder de Borland.

La estructura de clases que se empleó se representa esquemáticamente en este diagrama

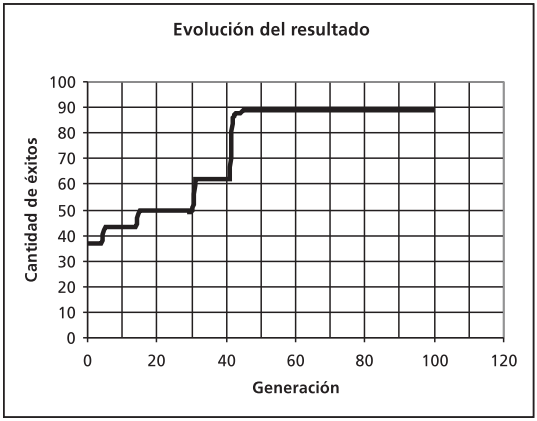


La clase Raíz era abstracta de manera que las particularidades de cada tipo de nodo se implementaban en las clases que de ella se derivaban.

Se empleó intensivamente las estructuras provistas por el C++ ANSI y, en particular, las STL (Standard Template Library)

Resultados

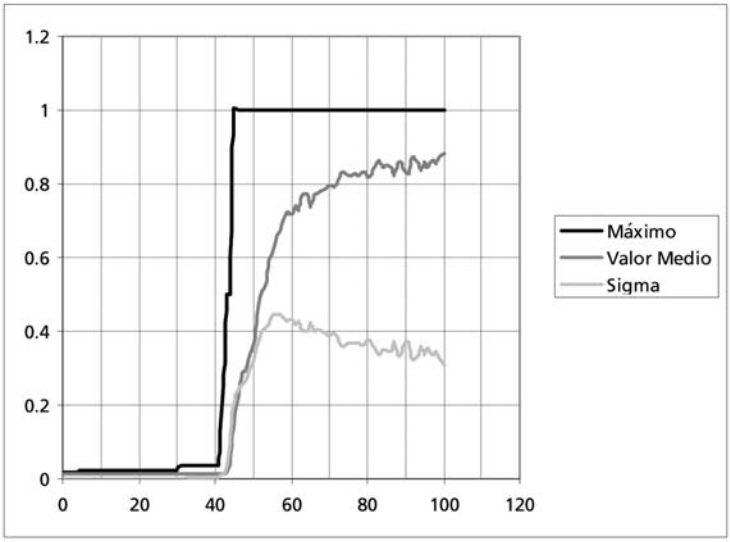
Usando una población inicial de 500 fórmulas y 100 generaciones, con una profundidad mínima de 4 y una profundidad máxima de 17 y con una longitud máxima de secuencia de 3 se obtuvieron los siguientes resultados:



De la curva se desprende:

- En la primer generación la mejor fórmula accede a 35 trozos de comida
- En la generación 44 se obtiene una fórmula que puede acceder a los 89 trozos de comida en 387 pasos

Si se gráfica la función de adaptación $\Phi(f) = 1/(1+Fa)$ para la mejor solución, su valor medio y varianza para cada generación, se pueden extraer las siguientes conclusiones:

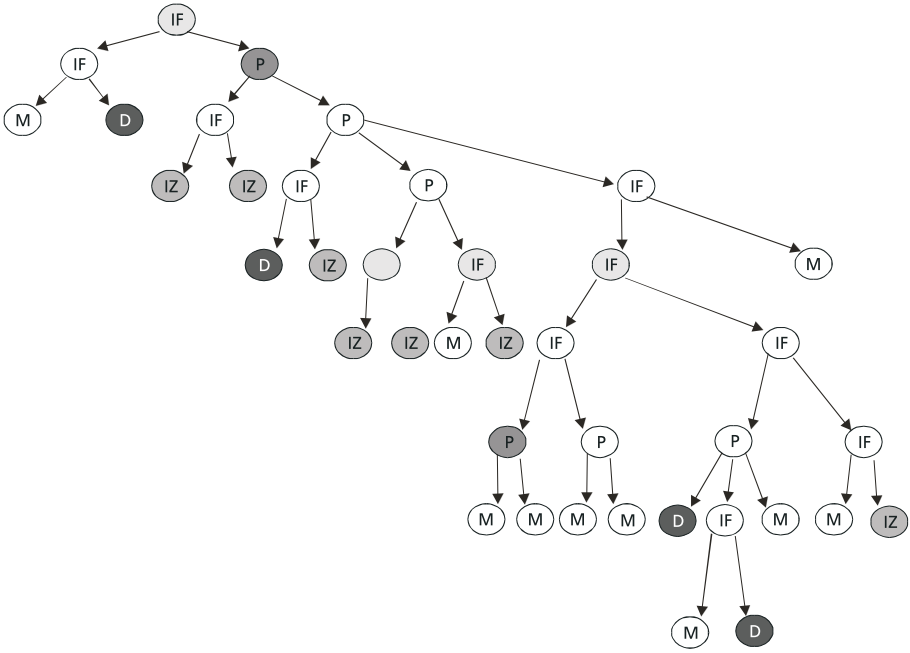


- El valor medio y la varianza permanecen muy bajos hasta la generación 44
- A partir de dicha generación todas las variables se incrementan abruptamente

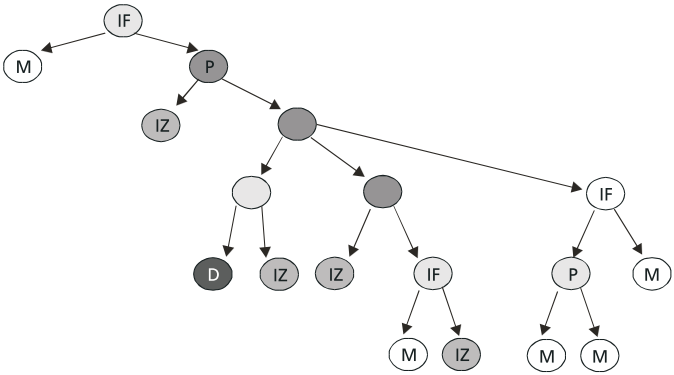
La solución obtenida es la siguiente:

(IfTenElse (IfTenElse (Mover) (Derecha)) (ProgN (IfTenElse (Izquierda) (Izquierda)) (ProgN (IfTenElse (Derecha) (Izquierda)) (ProgN (IfTenElse (Izquierda) (Izquierda)) (IfTenElse (Mover) (Izquierda))) (IfTenElse (IfTenElse (IfTenElse (ProgN (Mover) (Mover)) (ProgN (Mover) (Mover))) (IfTenElse (ProgN (Derecha) (ProgN (Mover) (Derecha)) (Mover)) (IfTenElse (Mover) (Izquierda)))) (Mover)))))

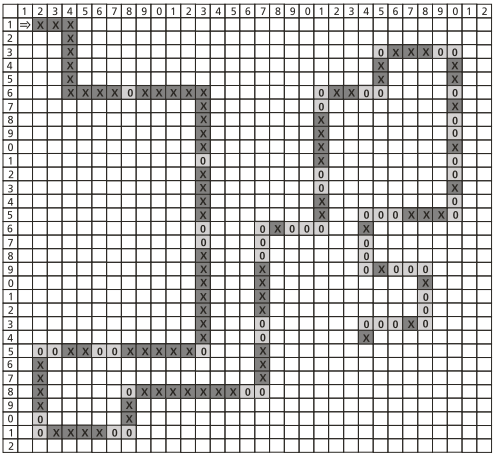
La misma puede representarse en forma de árbol de la siguiente manera



El árbol anterior puede simplificares dando por resultado el siguiente:



El camino seguido por la "hormiga" es:



Conclusiones

Las experiencias realizadas muestran que el camino que queda por recorrer es muy grande. En particular habría que:

- Realizar experiencias con diferentes generadores de números random para poder decidir si la sensibilidad de los resultados con respecto a la selección de la semilla es una deficiencia del método o del algoritmo de generación de números pseudo random. En principio se empleó la función rand() que devuelve un entero entre 0 y 32767. Para las pruebas realizadas se efectuaron más de 2.000.000 de llamadas a rand(). Ello significa que hubo una gran repetición de valores. Seria conveniente emplear una función de generación de números random que devolviera un long en lugar de un int.
- Probar diferentes distribuciones de probabilidad para la elección de las funciones (no terminales) y acciones (terminales).
- El aumento de la probabilidad de mutación no introduce cambios significativos en la dispersión de las soluciones. Este es un tema importante para investigar y desarrollar métodos que permitan incrementar dicha dispersión (Incremento de la variedad genética).
- Generar diferentes poblaciones que evolucionen independientemente y luego aplicar operadores de migración (Los mejores de una población reemplazan a los peores de otras poblaciones).
- Ensayar diferentes tipos de selección de padres
- Aplicar operadores de simplificación de las fórmulas
- Utilizar programas multithread para acelerar el proceso y, de esa manera, poder emplear poblaciones de mayor dimensión y mayor cantidad de generaciones.

Tanto los algoritmos genéticos como la programación genética constituye una fecunda fuente de investigación que puede ser aplicada, como lo fue en el caso de la programación de la producción de una planta de papel por el autor, en diferentes áreas de aplicaciones.