



Programación Genética

Semestre 2021-II



Contenido

- ◆ Antecedentes
- ◆ Definición
- ◆ Representación y dominio de aplicación
- ◆ Operadores Genéticos
- ◆ Teorema del Esquema en Programación Genética
- ◆ Generación de Módulos y Funciones Definidas Automáticamente (ADF's)
- ◆ Teoría de la Evolución Neutral
- ◆ Aplicaciones

Antecedentes

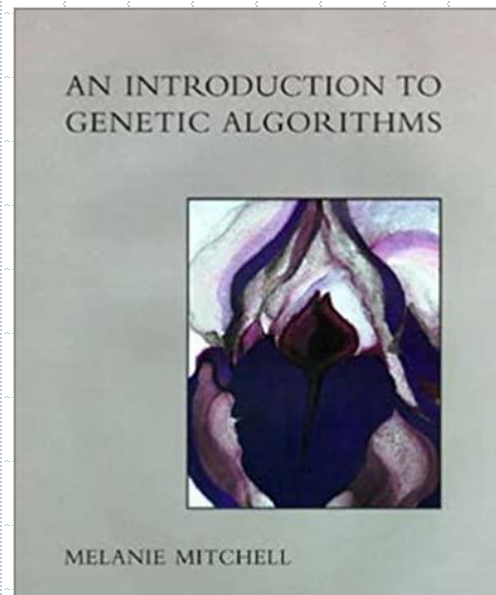
- ◆ Migrar de las implementaciones manuales: prog. estructurada, POO, librerías de objetos, etc. hacia una forma de prog. automáticamente
- ◆ Habilidad para que las computadoras aprendan a programar por sí mismas
- ◆ Desde los 50's, científicos han tratado de proporcionar a las computadoras la habilidad de aprender
- ◆ Samuel introdujo el término "machine learning" para nombrar este campo en 1959



Samuel, Arthur L. (1959). "Some Studies in Machine Learning Using the Game of Checkers". *IBM Journal of Research and Development*. **44**: 206–226.

Antecedentes

- ◆ El significado de este término fue “computadoras que programan por sí solas”
- ◆ Mitchell [1996] proporciona la siguiente definición: “Es el estudio de algoritmos de computadoras que mejoran automáticamente a través de la experiencia”

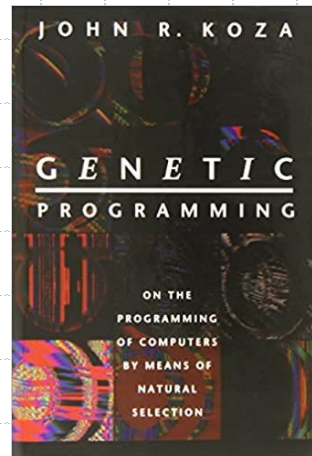


Antecedentes

- ◆ La Programación Genética (PG) tiene como objetivo precisamente este punto: inducir a una población de programas de computadora a mejorar automáticamente al ir experimentando con los datos sobre los cuales se están entrenando
- ◆ Por tanto, PG es parte de lo que se conoce como Machine Learning
- ◆ En la comunidad de ML, PG es un sistema de aprendizaje de máquina que evoluciona “estructuras de árbol”
- ◆ Crammer [1985] y Koza [1989] sugirieron el uso de una estructura de árbol como la representación de un programa
- ◆ Otros trabajos similares en la evolución de programas en LISP y PROLOG fueron presentados por Hicklin [1986], Fujiki y Dickinson [1987], Dickmanns et al. [1987]

Antecedentes

- ◆ Koza fue el primero en reconocer la importancia del método y demostrar su viabilidad para la programación automática [1989]
- ◆ En 1992 aparece el libro "Genetic Programming. On the Programming of Computers by Means of Natural Selection" por John Koza



Antecedentes (CE)

0	1	0	1	0	0	1	0
---	---	---	---	---	---	---	---



Binaria (GA)

4	8	1	3	9	5	6	12
---	---	---	---	---	---	---	----



Entera (GA)

0.01	1.53	-0.57	0.98	-1.06	-1.35	-0.77	1.08
------	------	-------	------	-------	-------	-------	------



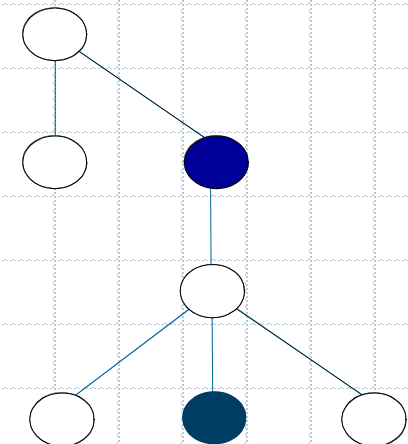
Real (GA, ES, EP)

4	12	1	3	9	0	0	0
---	----	---	---	---	---	---	---



Variable (GA)

Jerárquica-árbol (GP)



Antecedentes

Año	Atribuído a:	Técnica	Característica
1958	Friedberg	Aprendizaje de Máquina	Ensamblador virtual
1959	Samuel	Aprendizaje de Máquina	Término de ML
1965	Fogel, Owens y Walsh	Programación Evolutiva	Autómata
1965	Rechenberg, Schwefel	Estrategias Evolutivas	Vector de números reales
1975	Holland	Algoritmos Genéticos	Cadena de bits tamaño fijo
1978	Holland and Reitmann	Sistemas Clasificadores	Reglas
1980	Smith	Antecedentes de PG	Cadena de bits variable
1985	Cramer	Antecedentes de PG	Arbol
1986	Hicklin	Antecedentes de PG	LISP
1987	Fujiki and Dickinson	Antecedentes de PG	LISP
1987	Dickmanns, Schmidhuber and Winklhofer	Antecedentes de PG	Ensamblador
1992	Koza	Programación Genética	Arbol

Antecedentes

EC

Evolutionary
Computing

=

GA

Genetic Algorithms
(Holland, 1975)

+

ES

Evolution Strategies
(Rechenberg, 1973)

+

EP

Evolutionary Programming
(Fogel, Owens and Walsh, 1966)



GP

Genetic Programming
(Koza, 1992)

Referencias Antecedentes

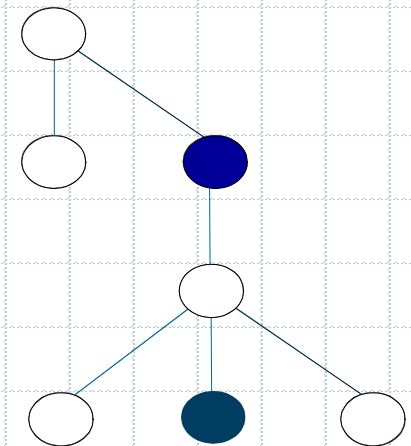
- ◆ **Cramer, N.L.** (1985) A Representation for the Adaptive Generation of Simple Sequential Programs. *In Proc. of Int. Conf. on Genetic Algorithms and the Applications* (Grefenstette, J.J., editor), pp. 183-187.
- ◆ **Dickmanns, D., Schmidhuber, J. and Winklhofer, A.** (1987) *Der Genetische Algorithmus: Eine Implementierung in Prolog*. Research Project, Technical University Munich, Munich.
- ◆ **Fogel, L.J., A.J. Owens and M.J. Walsh** (1966) *Artificial Intelligence Through Simulated Evolution*. Wiley Publishing.
- ◆ **Friedberg, R.** (1958) A Learning Machine, Part I. *IBM Journal of Research and Developments*, **2**, pp. 2-13.
- ◆ **Fujiki, C. and Dickinson, J.** (1987) Using the Genetic Algorithm to Generate LISP Source Code to Solve the Prisoner's Dilemma. *In Proc. of Int. Conf. on Genetic Algorithms and the Applications* (Grefenstette, J.J., editor), pp. 236-240.
- ◆ **Hicklin, J.F.** (1986) *Application of Genetic Algorithm to Automatic Program Generation*. Master's thesis, Dept. of Computer Science, University of Idaho.
- ◆ **Holland, J.** (1975) *Adaptation in Natural and Artificial Systems*, The MIT Press.

Referencias Antecedentes

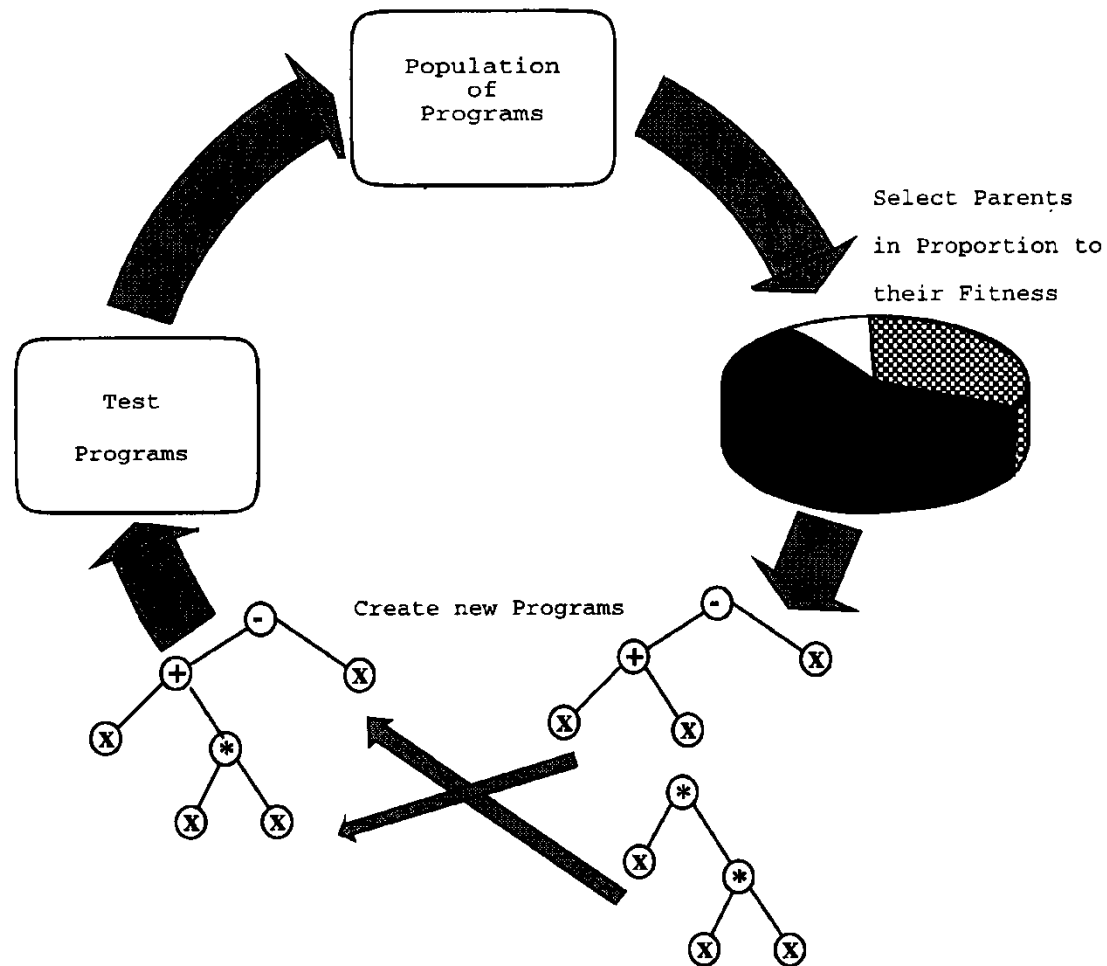
- ◆ **Holland, J. and Reitman, J.** (1978) Cognitive Systems Based on Adaptive Algorithms. In *Pattern-Directed Inference Systems* (Waterman, D. and Hayes-Roth, F., editors). Academic Press.
- ◆ **Koza, J. R.** (1989) Hierarchical Genetic Algorithms Operating on Populations of Computer Programs. In *Proc. of the 11th Int. Joint Conf. on Artificial Intelligence*, Vol. 1. Morgan Kaufmann, pp. 768-774.
- ◆ **Koza, J.R.** (1992) *Genetic Programming. On the Programming of Computers by Means of Natural Selection*, The MIT Press.
- ◆ **Mitchell, M.** (1996) *An Introduction to Genetic Algorithms*. The MIT Press.
- ◆ **Rechenberg, I.** (1965) *Cybernetic Solution Path of an Experimental Problem*. Ministry of Aviation, Royal Aircraft Establishment.
- ◆ **Samuel, Arthur L.** (1959). "Some Studies in Machine Learning Using the Game of Checkers". *IBM Journal of Research and Development*. **44**: 206–226.
- ◆ **Schwefel, H-P.** (1981) *Numerical Optimization of Computer Models*. John Wiley & Sons.
- ◆ **Smith, S.F.** (1980) *A Learning System Based on Genetic Adaptive Algorithms*. University of Pittsburgh.

Definición

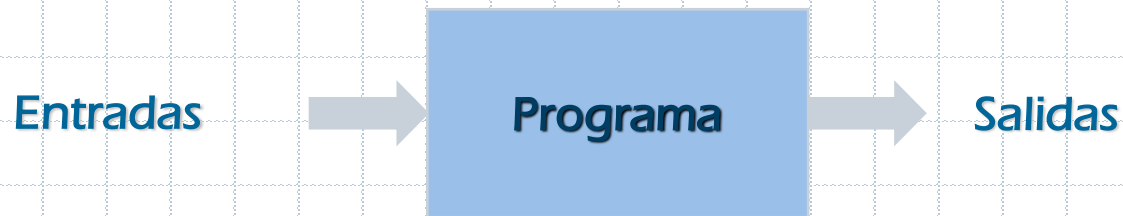
- ◆ Programación Genética (Koza, 1992; 1994) es una sub-clase de los populares Algoritmos Genéticos. En esta técnica evolutiva, cada individuo dentro de la población está constituido por funciones y argumentos codificados como árboles jerárquicos
- ◆ Esta característica de la programación Genética proporciona una manera de representación dinámica
- ◆ El espacio de búsqueda en PG es el espacio de todas las posibles estructuras compuestas de las funciones y argumentos definidos para el problema



Definición



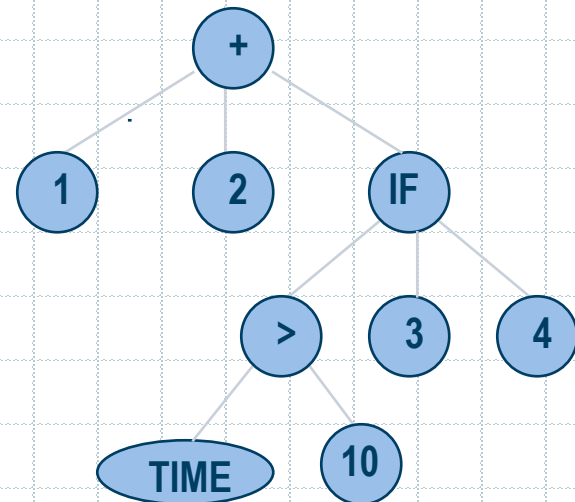
Definición



Programa = Arbol Jerárquico = Programa en LISP = Datos = Lista

Atomos (Terminales) = 1, 2, 10, 3, 4, TIME

Funciones = +, IF, >



Definición: Primitivas

Conjunto de Terminales. Conjunto de variables, constantes y funciones con cero entradas.

Conjunto de Funciones. Conjunto de operadores y funciones.

- Booleanas: AND, OR, NOT, XOR,
- Aritméticas: +, -, *, %,
- De relación: >, <, =,
- Trigonómicas, logarítmicas: SIN, COS, EXP, LOG,
- Condicionales: IF-THEN-ELSE, CASE, SWITCH,
- Iteraciones y Ciclos: DO-UNTIL, WHILE-DO, FOR-DO,
- Sub-rutinas o funciones de un dominio específico: MOVE-RANDOM, IF-FOOD-HERE,

Definición

Individuo Funciones + Terminales

$F = \{f_1, f_2, \dots, f_{NF}\}$ Conjunto de Funciones

Cada una de las funciones definidas en F toma un cierto número de argumentos (*arity*):

$\sigma(F) = \{\sigma(f_1), \sigma(f_2), \dots, \sigma(f_{NF})\}$

$T = \{a_1, a_2, \dots, a_{NT}\}$ Conjunto de Terminales (considerados también como funciones que toman cero argumentos)

Por tanto: $S = F \cup T$

El conjunto posible de estructuras (espacio de búsqueda) en PG es el conjunto de todas las posibles combinaciones de funciones que se pueden formar recursivamente a partir del conjunto de funciones y el conjunto de terminales definidos para el problema en estudio ...

(Koza, 1992, p.80)

Propiedades

En GP, una población de individuos representados en forma jerárquica deben satisfacer dos propiedades

Closure

Esta propiedad establece que toda función debe recibir argumentos de un solo tipo de dato y regresar valores que puedan ser argumentos de entrada de otra función. En otras palabras, cualquier función del conjunto de funciones, debe estar bien definida para cualquier combinación de argumentos que se encuentren.

Suficiencia

Basados en esta propiedad, la solución al problema en estudio puede ser formulada por medio de la composición de los conjuntos de funciones y terminales definidos previamente.

Crea Individuo

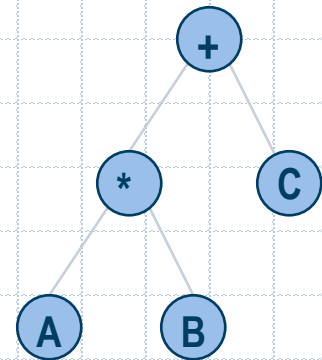
Conjunto de Terminales $T = \{A, B, C\}$

Conjunto de Funciones $F = \{+, -, *, \%, \text{IFTE}\}$

- Nodo raíz (función) con dos argumentos (+)

- Continua con función con dos argumentos


- Completar con terminales



El resultado es un programa
ejecutable válido sintácticamente

$(+ (* A B) C)$

Población Inicial

- ▶ Definir un límite máximo (e.j. Máximo número de nodos, máxima profundidad)
- ▶ Crear aleatoriamente un nodo raíz  elemento función
- ▶ Tomar en consideración el número de argumentos de la función (aridad) para la generación de estructuras válidas que satisfagan las propiedades de *closure* y *suficiencia*.
- ▶ Un elemento del conjunto de terminal seleccionado como un nodo en el árbol, es un nodo terminal y en este punto el proceso finaliza
- ▶ Tres métodos son principalmente utilizados en la creación de la población inicial en GP: “*full*”, “*grow*” y “*ramped half-and-half*”

Función Objetivo

- ▶ Es una medida del desempeño de un individuo (solución potencial) en el dominio del problema
- ▶ En muchos casos, el valor de fitness es definido por el error producido por el individuo (o programa) (e.j. Regresión simbólica)
- ▶ Generalmente, cada individuo es evaluado sobre un número de diferentes casos, los cuales se conocen como *fitness cases*. (e.j. En el dominio de regresión simbólica, el desempeño se mide sobre una serie de puntos y un valor de desempeño escalar puede darse por medio de la variancia de los residuos).
- ▶ Al igual que en GA, estrategias de asignación de *fitness* tales como scaling, ranking, y estrategias multi-objetivos como son el esquema Pareto, funciones de agregación, etc. pueden ser implementadas.

Operadores Genéticos

► *Reproducción*

Operador asexual

Similar a GA

Base de la supervivencia del más apto

Copia sin alteraciones del individuo
seleccionado

► *Cruzamiento*

Operador principal

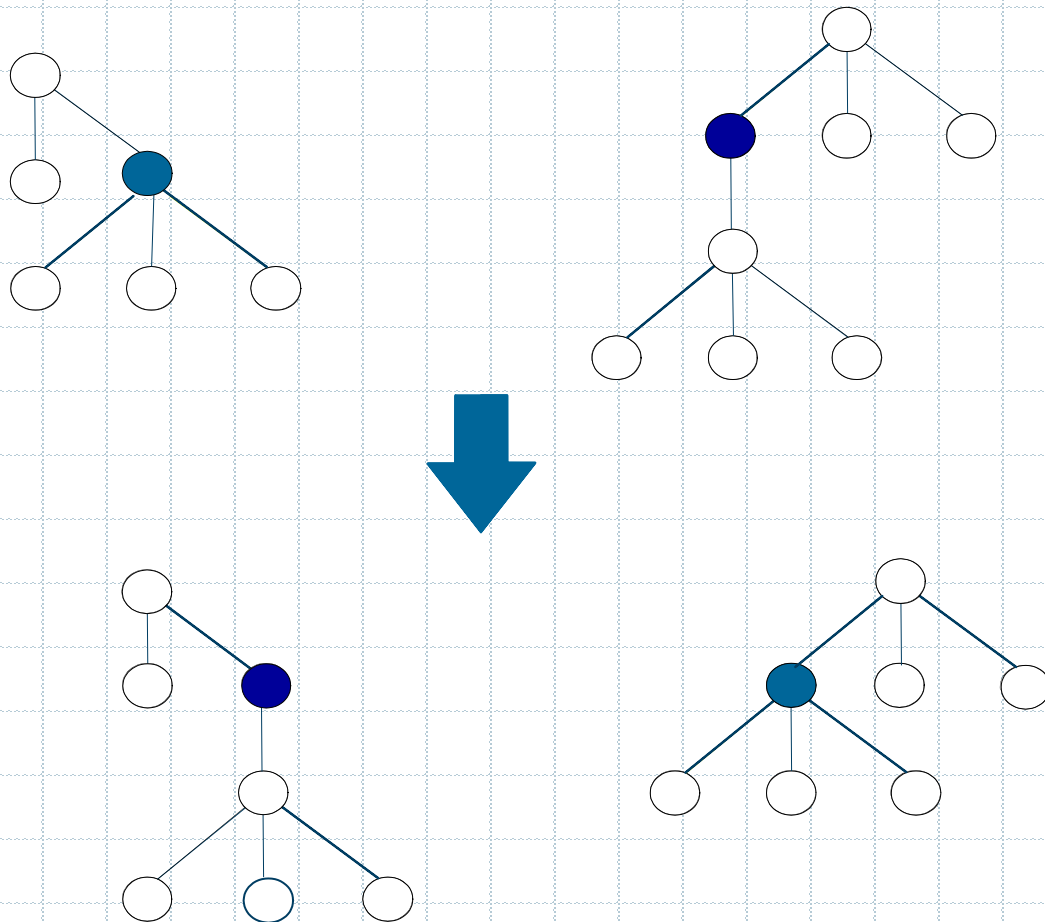
Operador sexual

► *Mutación*

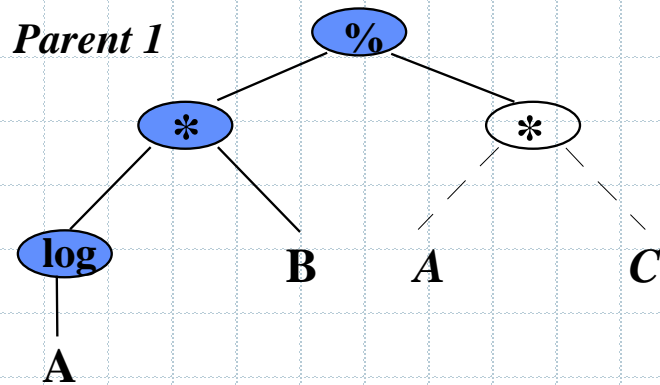
Operador secundario

Operador asexual

Cruza

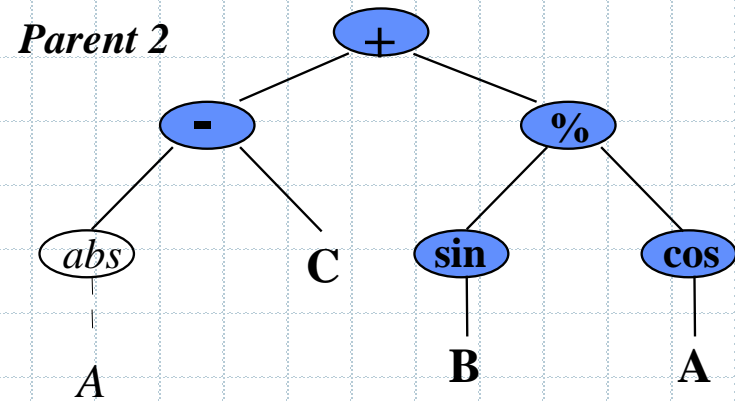


Cruza



$(\% (* (\log A) B) (* A C))$

$$\frac{B \log(A)}{AC}$$



$(+ (- (abs A) C) (\% (\sin B) (\cos A)))$

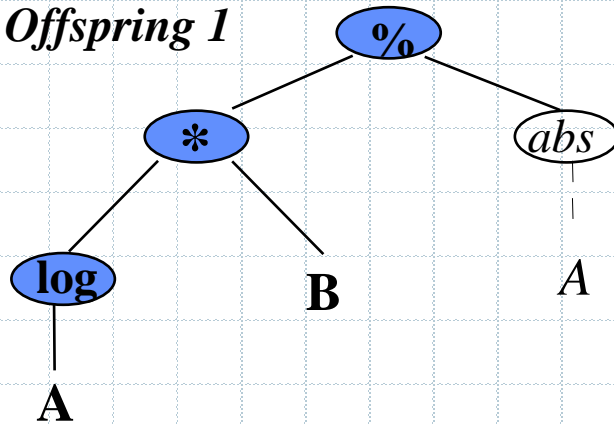
$$abs(A) - C + \frac{\sin(B)}{\cos(A)}$$

Genera dos nuevos individuos árboles que heredan características de ambos padres.

- Selección aleatoria de un nodo en cada uno de los padres.

Cruza

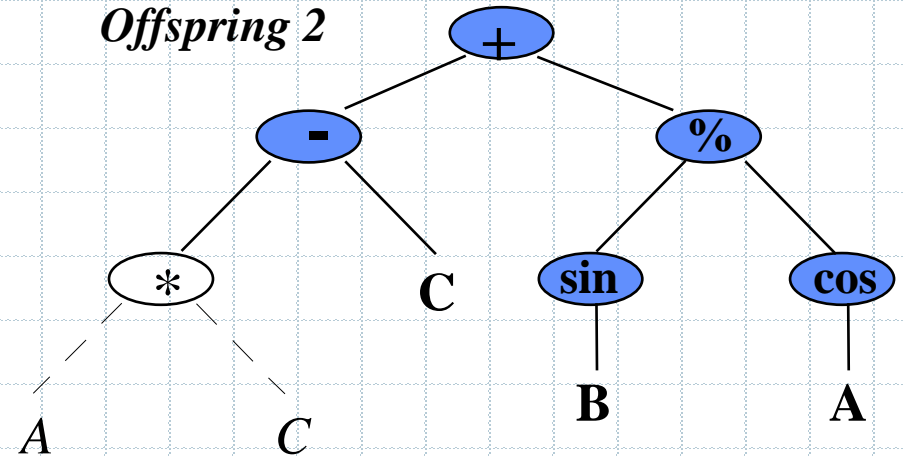
Offspring 1



$(\% \ (* \ (\log A) \ B) \ (abs \ A))$

$$\frac{B \log(A)}{abs(A)}$$

Offspring 2



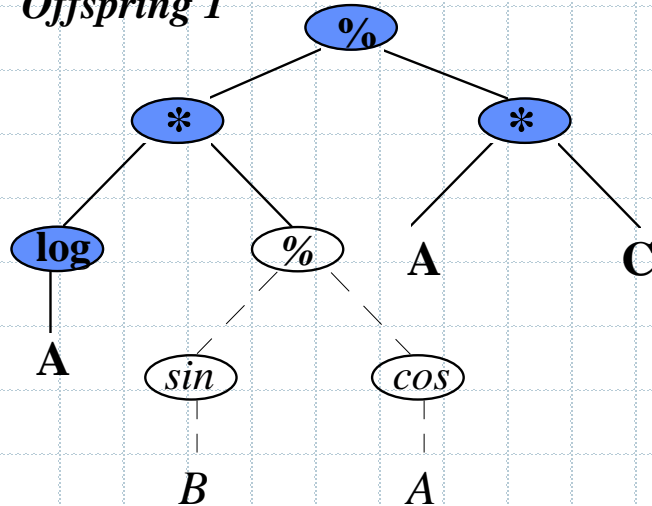
$(+ \ (- \ (* \ A \ C) \ C) \ (\% \ (\sin B) \ (\cos A)))$

$$AC - C + \frac{\sin(B)}{\cos(A)}$$

- Los sub-árboles generados a partir del nodo de cruzamiento son intercambiados entre los padres.

Cruza

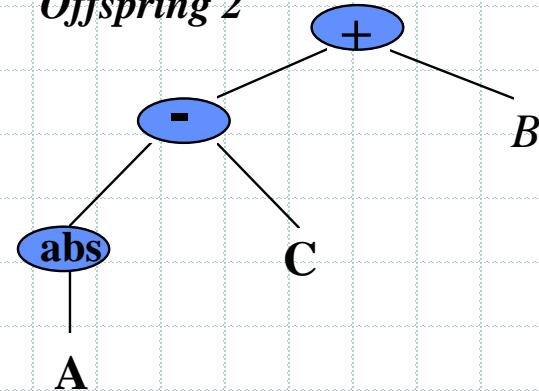
Offspring 1



$(\% (* (\log A) (\% (\sin B) (\cos A)))) (* A C)$

$$\frac{\log(A) \frac{\sin(B)}{\cos(A)}}{AC} = \frac{\log(A) * \sin(B)}{AC \cos(A)}$$

Offspring 2



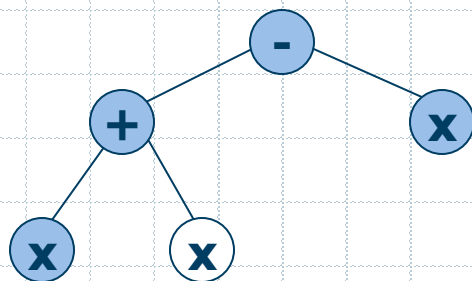
$(+ (- (\text{abs } A) C) B)$

$\text{abs}(A) - C + B$

GP-cruzamiento eligiendo un nodo terminal y un nodo interno como nodos de cruzamiento.

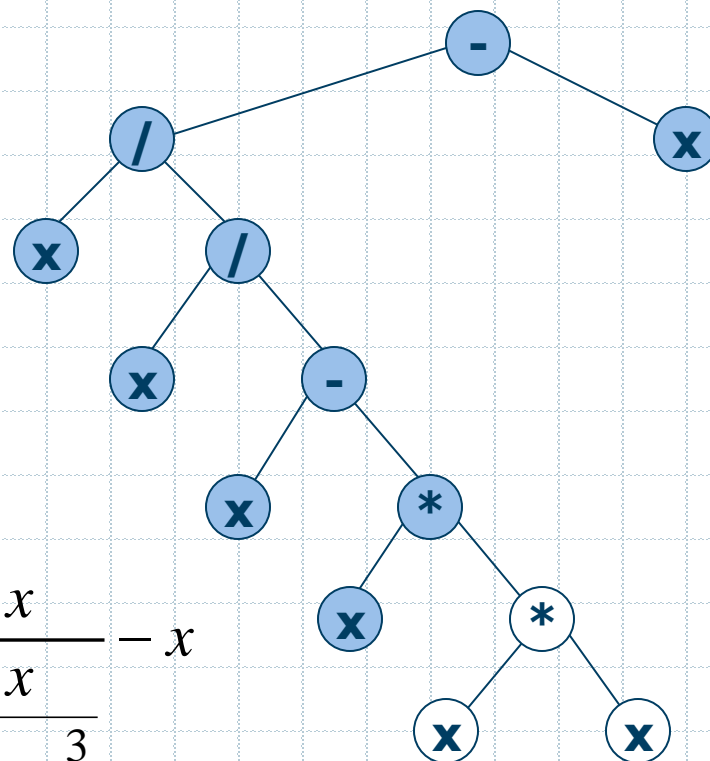
Cruza

Padre



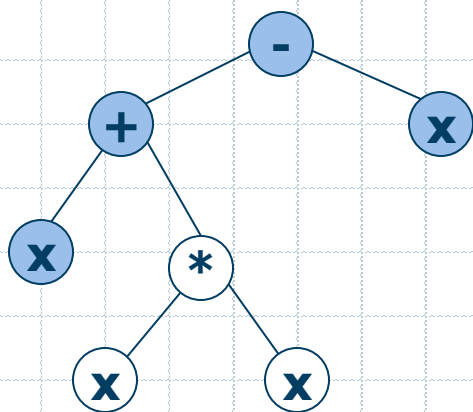
$$2x - x$$

Madre

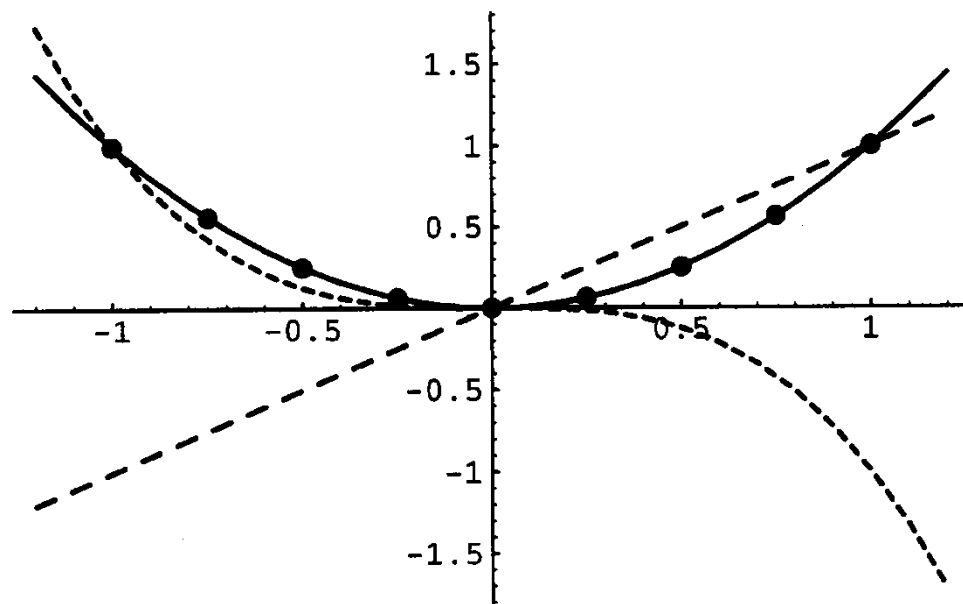


$$\frac{x}{x} - x$$
$$x - x^3$$

Cruza



$$x + x^2 - x$$



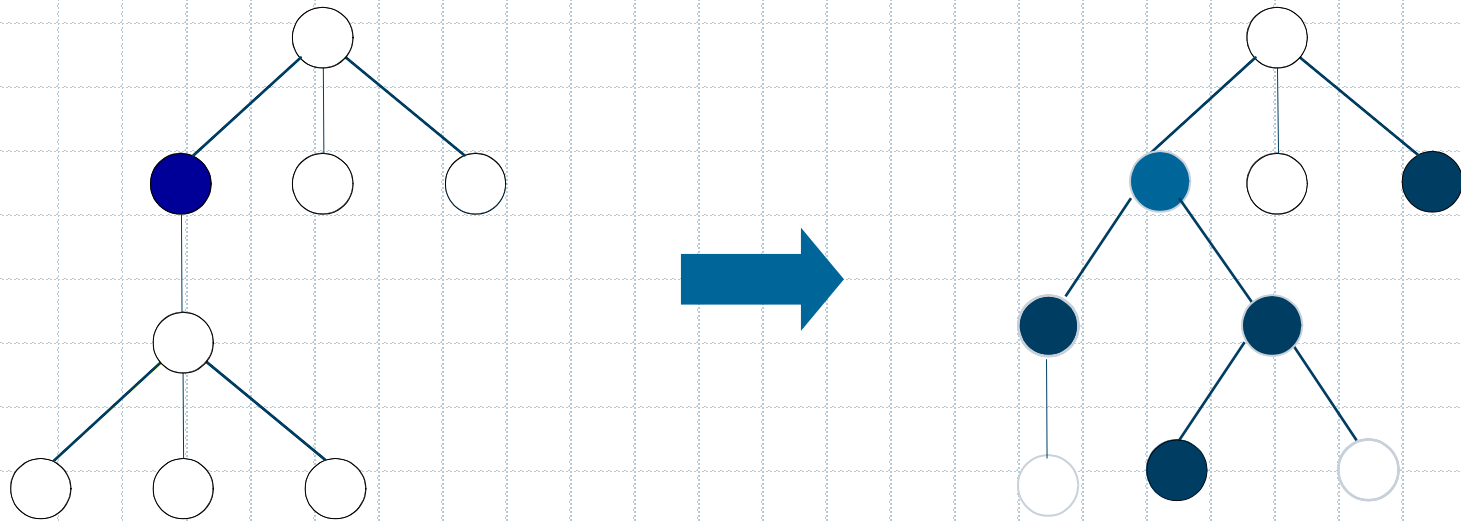
—————

- - - - -

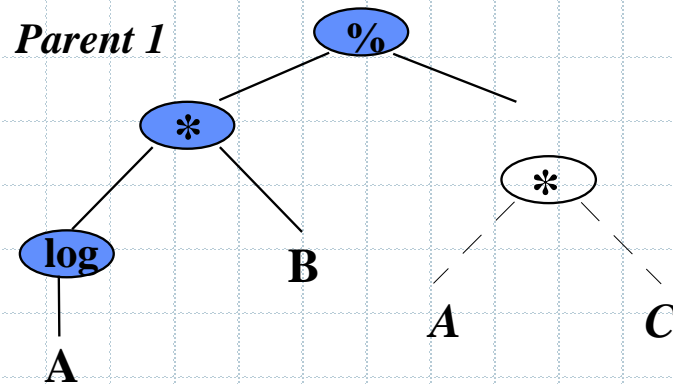
.....

x²
Padre
Madre

Muta

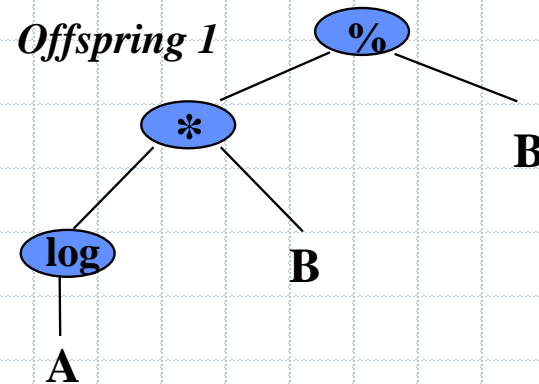


Muta



$(\% (* (\log A) B) (* A C))$

$$\frac{B \log (A)}{AC}$$

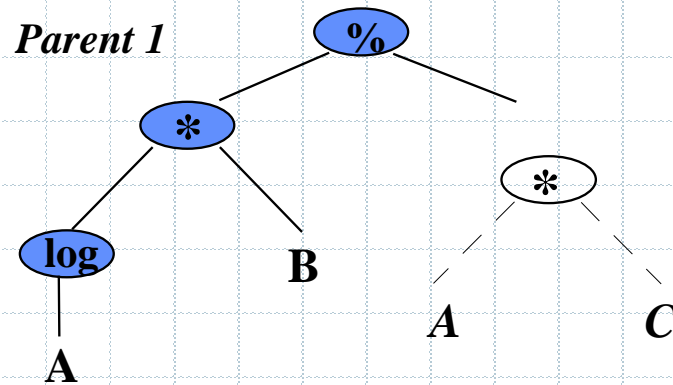


$(\% (* (\log A) B) B)$

$$\frac{B \log (A)}{B}$$

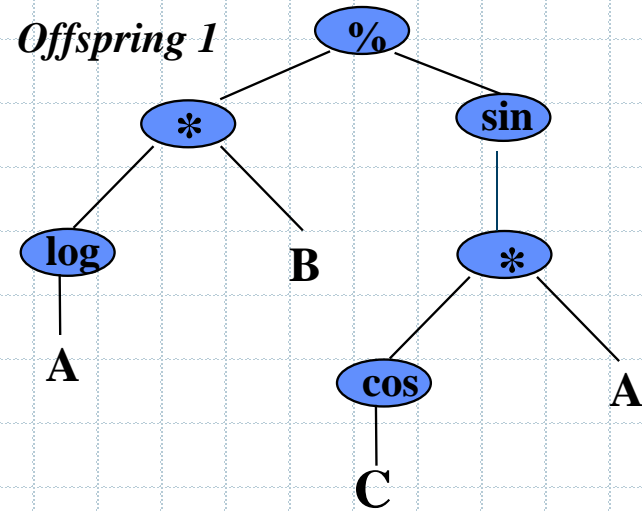
- Selecciona aleatoriamente un nodo.
- El sub-árbol asociado a este nodo es substituido por un nuevo sub-árbol generado aleatoriamente a partir del conjunto de funciones y terminales.

Muta



$(\% \ (* \ (\log A) \ B) \ (* \ A \ C))$

$$\frac{B \log(A)}{AC}$$



$(\% \ (* \ (\log A) \ B) \ \sin) \ (* \ A \ \cos(B)) \)$

$$\frac{B \log(A)}{\sin(A \cos C)}$$

Referencias

- ◆ **Banzhaf, W., P. Nordin, R.E. Keller and F.D. Francone (1998)** *Genetic Programming: An Introduction*. Morgan Kaufmann Publishers.
- ◆ **Koza, J.R.** (1992) *Genetic Programming. On the Programming of Computers by Means of Natural Selection*, The MIT Press.
- ◆ **Koza, J.R.** (1994) *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.

Cruza

- ***Recombinative Guidance***

(Iba and de Garis, 1996)

Cruzamiento adaptativo

Control de los sub-árboles → elección de los nodos de cruzamiento

Valores de fitness de cada “bloque” (sub-árbol) son considerados para seleccionar el punto de cruzamiento

- ***Selective Self-Adaptive Crossover Self-Adaptive Multi-Crossover***

(Angeline, 1996)

Mecanismo similar a (Iba and de Garis, 1996)

Mutación

- ▶ **Grow**

Substitución de un punto terminal por un sub-árbol

- ▶ **Shrink**

Substitución de un sub-árbol por un nodo terminal

- ▶ **Cycle**

Substitución de un nodo interno (función) por una nueva función

- ▶ **Switch**

Intercambio de dos sub-árboles seleccionados aleatoriamente en el mismo padre

Mutación

► *Numerical Terminal*

Alteración de un nodo terminal (valor numérico, no terminal)

► *Substitución de Nodo*

Similar a la mutación binaria en AG. Substituye el contenido de un nodo seleccionado aleatoriamente por otra primitiva considerando el número de argumentos de la función contenida en el nodo a substituir

Teorema del Esquema en PG

- ◆ Bases: Teorema del Esquema de Holland (1975)
- ◆ Teorema del Esquema en AG:
 - Binario
 - Esquema (H) es una plantilla de similitudes
 - Considera en el alfabeto el elemento * (don't care)
 - Orden ($o(H)$) es el número de posiciones con valores fijos (1 o 0)
 - Longitud ($\ell(H)$) es la distancia entre los dos símbolos no-* más lejanos
 - Holland obtuvo un resultado el cual predice cómo el número de cadenas en una población pertenecientes a un cierto esquema se espera varíe de una generación a otra bajo los efectos de la selección (proporcional), mutación y cruza en un solo punto.

Teorema del Esquema en PG

$$m(H, t+1) \geq m(H, t) \frac{f(H)}{\bar{f}} \left[1 - p_c \frac{\delta(H)}{\ell - 1} - p_m o(H) \right]$$

f	fitness
H	esquema
m	número de esquemas
δ	longitud
o	orden
p_c	probabilidad de cruzamiento
p_m	probabilidad de mutación
ℓ	longitud de la cadena

Teorema del Esquema en PG

Del esquema en AG al esquema en PG

- ◆ Esquema en AG $\rightarrow \{0, 1, *\}$
- ◆ Representación equivalente
 - Conjunto de pares $[(c_1, i_1), (c_2, i_2), \dots]$
 - c_j son sub-cadenas de caracteres del alfabeto original las cuales representan grupos de símbolos continuos definidos (componentes del esquema)
 - i_j proporciona la posición de las c_j 's
 - E.j.: $* 1 0 * 1$
 - ◆ 2 componentes expresados como $[(1\ 0, 2), (1, 5)]$
 - $* * 1 1 * *$ $\rightarrow [(1\ 1, 3)]$
 - $1 1 * 1 1 1 * 1$ $\rightarrow [(1\ 1, 1), (1\ 1\ 1, 4), (1, 8)]$

Teorema del Esquema en PG

- ◆ Un esquema es destruido por el operador de cruza cuando uno o mas de sus componentes no se transmiten a su descendiente o son parcialmente destruidas
- ◆ ***Hipótesis de Bloques Constructores*** (Holland, 1975; Goldberg, 1989)
 - “ AG's trabajan al combinar esquemas cortos y relativamente aptos para formar soluciones completas ”
- ◆ Con base a la representación alternativa de pares:
 - “ Cruza concatena los componentes de esquemas de orden pequeño para formar esquemas de mayor orden ”
- ◆ TEAG: “Descripción de la variación del número de ciertos grupos de bits en la población (componentes c_j en la posición i_j) ”

Teorema del Esquema en PG

- ◆ El número de cadenas en la población t del esquema H , $m(H, t)$, es igual al número de veces los componentes c_1, \dots, c_n están presentes simultáneamente en las cadenas de la población. Esto se denomina **instanciaciones** $i(H, t)$.

◆ E.j.:

1	1	0	0	1
0	0	0	1	1
0	0	0	0	1
1	1	0	1	1

- $m(1\ 1\ *\ *\ *, t) = m([(1\ 1, 1)], t) = i([(1\ 1, 1)], t) = 2$
- $M(*\ * \ 0\ *\ 1, t) = m([(0, 3), (1, 5)], t) = i([(0, 3), (1, 5)], t) = 4$

Teorema del Esquema en PG

- ◆ Qué ocurre si se omite la posición?
- ◆ *El número de cadenas pertenecientes a un esquema dado es ahora diferente al número de instanciaciones del esquema*
- ◆ $m([(1\ 1)], t) = 3$ $i([(1\ 1)], t) = 4$
- ◆ $m([(0), (1)], t) = 4$ $i([(0), (1)], t) = 20$
- ◆ Cuántos individuos tienen un cierto esquema?
- ◆ Cuántos esquemas hay en la población?

Teorema del Esquema en PG

- ◆ Definiciones de esquema que no consideran la posición pueden ser útiles en representaciones más complejas y funciones de fitness independientes (problemas de opt. combinatoria)
- ◆ Considerar o no la posición en el caso de la PG es relevante
- ◆ Primero trabajos en T.E.P.G. no consideran la posición, concentrándose en el análisis de la propagación de los **componentes** en la población más que en la manera en la cual el número de programas con un cierto esquema cambia en el tiempo

TEPG

- ◆ Cuál es la definición de esquema en PG útil en la formulación para la descripción del espacio de búsqueda?
- ◆ Esquemas son simplemente una forma de definir sub-conjuntos de espacios de búsqueda
- ◆ En PG, diferentes formas de particionar el SS (e.j. de acuerdo a las funciones, fitness, número de nodos en el árbol, altura de los árboles)
- ◆ Seguir la lógica de la definición de esquema en AG

TEPG

- ◆ Koza [1992, pp. 117-118] primero en intentar explicar por qué la PG trabaja
 - “ Un esquema en PG es el conjunto de todos los individuos árboles de la población que contienen, como subárboles, uno o más árboles especificados. Un esquema es un conjunto de S-expresiones en LISP compartiendo ciertas características en común ”
- ◆ **H** es un conjunto de sub-programas
- ◆ E.j. $H = [(+ 1 x), (* x y)]$ representa todos los programas que contienen al menos una ocurrencia del esquema $(+ 1 x)$ y al menos una del esquema $(* x y)$
- ◆ El esquema $H = [(x)]$ puede ser instanciado en dos forma en el programa $(+ x x)$

TEPG

- ◆ La definición de esquema de Koza implica que ningún esquema es definido por una S-expresión especificada incompleta tal como $(+ \# 2)$, donde $\#$ es un comodín
- ◆ Se tienen comodines en la definición de Koza pero se restringe su uso a S-expresiones las cuales incluyen el esquema más que usarlas dentro de los esquemas
- ◆ Una definición más general de esquema considera la colección de S-expresiones definida completamente así como S-expresiones incompletas (considera $\#$ dentro del esquema)
- ◆ La definición de esquema no especifica exactamente como los fragmentos o S-expresiones están ligados en una instancia pero se requiere que se tenga una ocurrencia de estos

TEPG

◆ Considerar el esquema $H = [(+ 3 4), (+ 3 4), (- \# \#)]$ y los siguientes tres programas:

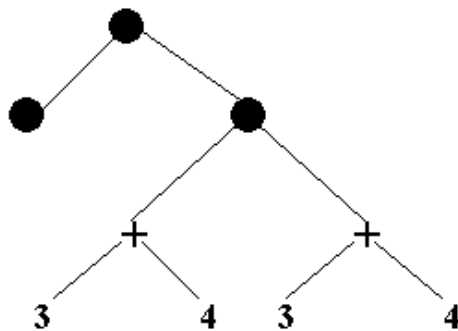
- (IF (+ 3 4) (+ 3 4))- x 2))
- (IF (- x 2) (+ 3 4) (+ 3 4))
- (AND (+ 3 4) (+ 3 4) (+ 3 4) (- x y))

Cuántas instancias $i(H, t)$ se tienen del esquema H en cada uno de los individuos?

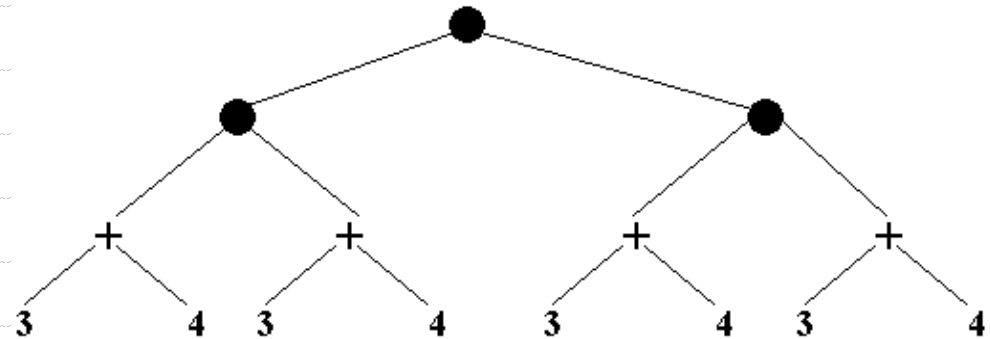
◆ Una definición más general de esquema es un conjunto de pares los cuales consideran los fragmentos o S-expresiones completas con el número de ocurrencias de cada una que debe cumplir una instancia. E.j. $H = [((+ 3 4), 2), ((- \# \#), 1)]$

TEPG

- ◆ Definición de Esquema de O'Reilly [1995]:
 - “ Un esquema **H** en PG es un conjunto de pares. Cada par es una S-expresión única o fragmento y un entero que especifica cuántas instancias de la S-expresión o fragmento comprende **H**”
- ◆ En estas definiciones de esquemas no se considera la posición y el concepto de instanciación es por tanto relevante
- ◆ E.j. Dado el esquema $H = [((+ 3 4), 2)]$, cuántas instancia de H se tiene en los siguientes programas:



(i)



(ii)

TEPG

- ◆ El programa (+ 4 3) no concuerda con el esquema $H=[(+ 3 4), 1]$
- ◆ Un programa instancia a un esquema una vez para cada forma el esquema encaja en el programa. El interés se enfoca, por tanto, a contar el número esperado de ocurrencias de un patrón de programa y un patrón puede ocurrir más de una vez en un programa.
- ◆ Definición:
Considerar un programa h . Una instanciación de un esquema H por un programa h , $inst(h,H)$, es un elemento de $Inst(h,H)$, el cual es una función que produce un conjunto de todas las ocurrencias de H en el programa h .

La notación $inst(h,H)$ no especifica la posición dentro del programa.

TEPG

- ◆ Orden de un esquema en PG:

- “ Es el número de nodos correspondientes a S-expresiones y fragmentos con valores definidos “

- Esquemas de mayor orden presentarán pocas instancias en una población en comparación con esquemas de orden menor.

- ◆ E.j. $H=[((+ 3 4), 2)]$ es de orden 6, $H=[((IF \# \# \#), 1)]$ es de orden 1 y $H=[((IF a \# \#), 1)]$ es de orden 2.

- ◆ La longitud D de una instanciación del esquema en PG es la suma de sus longitudes fijas y variables:

$$D(inst(h,H), H) = D_{fixed}(H) + D_{var}(inst(h,H), H)$$

TEPG

- ◆ La longitud fija de un esquema H , $D_{fixed}(H)$, es el número de ligas en una S-expresión o fragmento, sin considerar ligas conectadas a comodines.
- ◆ E.j. en el esquema $H=[(+ 3 4), 2]$ del ejemplo (i) e (ii) , $D_{fixed}(H) = 4$.
- ◆ La longitud variables de una instancia del esquema, $D_{var}(h, H)$, es el número de ligas las cuales se conectan a S-expresiones o fragmentos en H , e.j. la suma de las longitudes de las rutas más cortas entre un fragmento de esquema y el antecesor común de todos los fragmentos de esquema en la instancia. $D_{var}(h, H)$ debe ser calculado para cada instancia y depende de como la instancia del esquema es considerada en el programa.

TEPG

- ◆ La ruptura de una instanciación ***inst(h,H)*** de esquema ocurren cuando un nodo en h es seleccionado como punto de cruce y el intercambio de sub-árbol con un sub-árbol de otro programa cambia h tal que no se instancia ***H***.
- ◆ Si el espacio de corte de una instanciación de esquema es el número de nodos del programa h , $Size(h)$, y $P_d(h, H) = P_d(inst(h,H), H)$, la probabilidad de ruptura bajo cruce de una instanciación del esquema es su longitud dividida por el número de nodos de h :

$$P_d(h, H) = \frac{D_{fixed}(H) + D_{var}(h, H)}{Size(h)}$$

TEPG

- ◆ Si se considera un sesgo para la selección del nodo de cruza (función o terminal), $P_d(h, \mathbf{H})$ se expresa ahora como:

$$P_d(h, H) = \frac{L_b V(H) + (1 - L_b)(D(h, H) - V(H))}{Size(h)}$$

donde L_b es la probabilidad de seleccionar un nodo hoja (terminal) y $1 - L_b$ es la probabilidad de selección de un nodo interno. El número de hojas de un esquema \mathbf{H} se denota como $V(\mathbf{H})$ y $D(h, \mathbf{H}) = D_{fixed}(\mathbf{H}) + D_{var}(h, \mathbf{H})$.

TEPG

- ◆ La compactibilidad de una instanciación de esquema es el complemento de su probabilidad de ruptura: $c(h) = 1 - P_d(h, \mathbf{H})$.

TEPG-O'Reilly

- ◆ TEPG expresada el número esperado de instancias de un esquema
- ◆ Retomando el TEAG para cadenas binarias de longitud fija, expresa el número esperado de un esquema en la población del tiempo t al $t+1$. Se presentan tres factores:
 - El número esperado de un esquema en el tiempo t en una población de n cadenas
 - El factor reproductivo del número esperado de un esquema por efectos de la selección proporcional
 - La probabilidad de que un esquema sobreviva a la cruce y mutación

TEPG-O'Reilly



Para formular en forma similar el TEPG, se tienen los siguientes ajustes y requerimientos:

1. El TEAG estima el número esperado de un esquema. TEPG considera el número esperado de instancias de un esquema debido a que un programa puede instanciar un esquema dado más de una vez. Definiendo $i(H,t)$ como el número de instancias de un esquema H en el tiempo t en la población de programas. $E[i(H,t)]$ denota el número esperado de instancias de un esquema H .
2. El número esperado de instancias de un esquema H reproducidos por medio de selección proporcional de un programa puede calcularse como:

$$E[i(H,t)] = i(H,t) \frac{\hat{f}(H,t)}{\bar{f}(t)}$$

TEPG-O'Reilly

Donde

$$\hat{f}(H, t)$$

fitness promedio del esquema H en el tiempo t.

$$\bar{f}(t)$$

fitness promedio de los programas en la población

El fitness promedio de un esquema H es la suma de las contribuciones parciales de los fitness de los programas que instancian H. La contribución de fitness de un programa es proporcional al número de instanciaciones que el programa adiciona a las instanciaciones totales de H en la población.

TEPG-O'Reilly

- ◆ Debido a que la PG standard [Koza, 1992] no usa mutación, no se considera en esta propuesta de TE. Se define la estimación de la probabilidad de que una instancia H se pierda en función de $P_d(h, H)$ la cual no es la misma para cada instancia que involucre H . La probabilidad de que una instancia de H se pierda está definida por

$$P_d(H, t) = P_d(h, H)$$

TEPG-O'Reilly

$$E[i(H, t + 1)] \geq i(H, t) \frac{\hat{f}(H)}{\bar{f}(t)} (1 - P_{xo} P_d(H, t))$$

donde

$$E[i(H, t)]$$

número esperado de instancias de H en t

$$\hat{f}(H, t)$$

fitness promedio observado de las intancias del esquema H

$$\bar{f}(t)$$

fitness promedio de la población

$$P_d(H, t)$$

probabilidad de pérdida de un esquema

$$P_{xo}$$

probabilidad de cruza

Bloques Constructores

Bloques Constructores: Esquemas de orden bajo, compactos con un desempeño observado por encima del promedio que se espera sean muestreados con rangos que en generaciones futuras se incrementa y/o crece exponencialmente

Hipótesis de Bloques Constructores: Establece que PG combina BC de orden bajo, compacta soluciones parciales altamente aptas de muestras pasadas, para formar individuos, que a lo largo de las generaciones, mejoran en su fitness

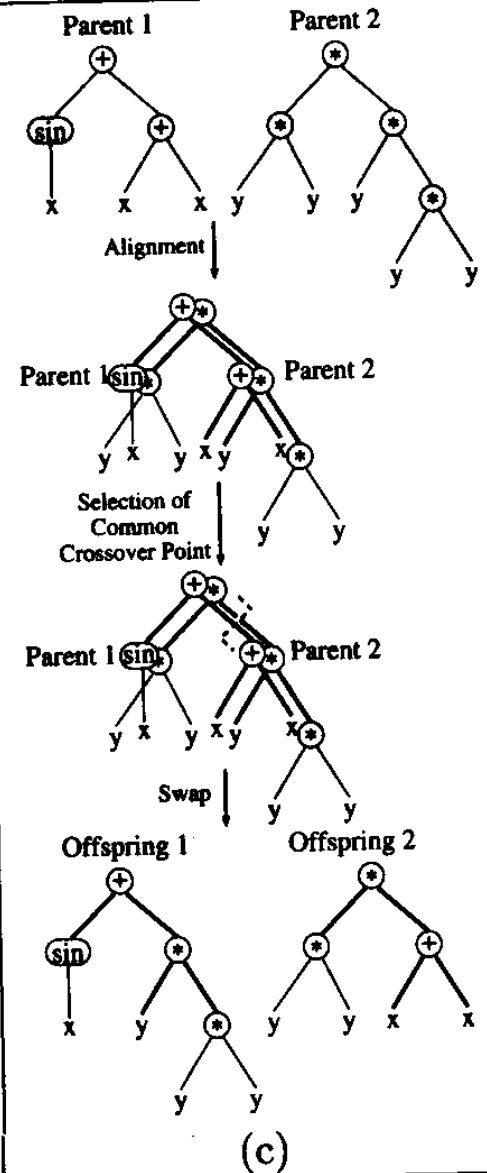
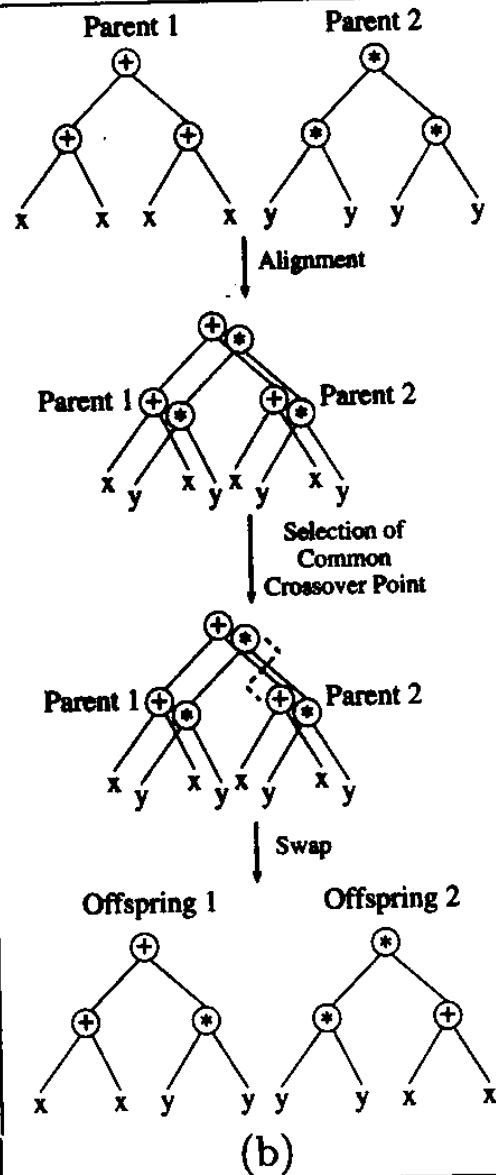
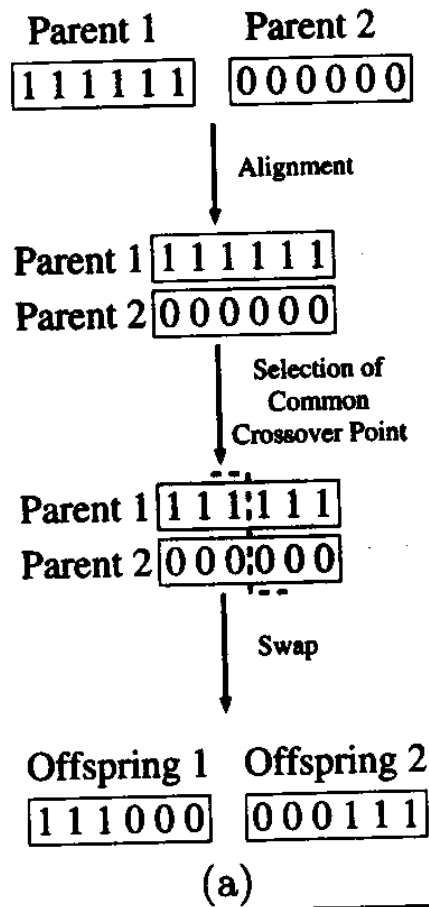
TEPG

◆ Primeros trabajos:

- Koza [1992] → Esquemas formados por expresiones completas
- Altenberg [1994] → Modelo probabilístico de PG, puede considerarse como la primera formulación matemática de TEPG
- O'Reilly [1995] → Punto de partida TEPG de Koza, considera comodines dentro de los esquemas (expresiones completas y fragmentos)
- Whigham [1996] → Derivó TEPG para su sistema de PG basado en gramáticas libre de contexto

TEPG

- ◆ Rosca [1997] → Basado en la idea de esquemas como componentes en un programa pero también definen sub-conjuntos reales del espacio de búsqueda. Considera esquemas con raíces de contenido definido. Ej. $H = (+ \# x)$
- ◆ Poli y Langdon [1997] → Esquemas de forma y contenido fijos. Consideran muta y cruza en un solo punto.



Convergencia y Bloat

- ◆ Convergencia: la población contienen sustancialmente individuos similares (EA's)
- ◆ En pocas ocasiones se maneja para decir que el algoritmo no progresa o más aun, ha terminado
- ◆ Convergencia en GP es más difícil de cuantificar que en AG
 - Distancia de Hamming no puede aplicarse directamente a estructuras variables
 - Medirla en términos de fenotipo?
- ◆ Mediciones tradicionales de diversidad genotípica (fuertemente relacionados por):
 - Pueden tener ancestros comunes
 - Árboles similares
 - Muchos mapean al mismo fenotipo

Bloat

- ◆ Rápido crecimiento de los individuos
- ◆ El crecimiento de los programas no necesariamente significa mejora en el fitness
- ◆ El bloat depende de la implementación de la PG, pero se argumenta que este crecimiento es exponencial
- ◆ El bloat ocurre no solo en PG sino en toda técnica de búsqueda que involucre representación variable
- ◆ Variaciones en el fitness es necesario para el bloat
- ◆ Mecanismos para controlar el bloat:
 - Tamaño y profundidad límites
 - Preferencia por individuos pequeños (complejidad) en el proceso de selección (doble elitismo)
 - Operadores genéticos “especiales”
 - Edición de código
 - Modularidad

Intrones y Bloat

- ◆ 1994 Peter Angeline
 - Muchas de las soluciones del libro de Koza contenían segmentos de código que podían removerse sin alterar el resultado de la solución
 - E.g. $b = a + 0$, $b = b * 1$
- ◆ Este código emerge espontáneamente como resultado del tipo de estructuras que PG maneja
- ◆ Importante para la evolución exitosa
- ◆ Primero en relacionar la emergencia de código "extra" con el término biológico de *intron*

Intrones y Bloat

◆ 1994 Tackett

- Observó que las ejecuciones de PG tienden al "bloat" (los individuos de la población aparentemente crecen en forma incontrolada hasta alcanzar el límite máximo permitido)
- El bloat en PG es ocasionado por la presencia de intrones
- 40% al 60% de todo el código de la población corresponden a intrones
- Bloat es un problema importante en PG, una vez que el crecimiento exponencial de intrones asociado con el bloat ocurre, casi siempre PG se estanca

Intrones y Bloat

- ◆ Angeline sugirió que la presencia de intrones tiene un efecto benéfico en la evolución
- ◆ Intrones efecto benéfico al inicio y etapa intermedia en el proceso evolutivo, protege los buenos bloques constructores
- ◆ Los intrones no tiene un efecto directo en el genotipo
- ◆ Tanto los intrones biológicos como artificiales (PG) no se traducen en el fenotipo

Intrones y Bloat

- ◆ Un intron es una característica del genotipo que emerge del proceso evolutivo de estructuras de longitudes variables
- ◆ Un intron no afecta directamente la supervivencia de los individuos en PG
- ◆ Ejemplo de intrones:
 - (NOT (NOT X))
 - (AND ... (OR X X))
 - (+ ... (- X X))
 - (+ X 0)
 - (* X 1)
 - (* ... (DIV X X))
 - (Move_left Move_right)
 - (IF (2=1) ... X)

Teoría de la Evolución Neutral

- ◆ Selección natural: teoría que explica la existencia de adaptación en la naturaleza
- ◆ Pero no es la única fuerza que conduce a la evolución
- ◆ A nivel molecular, se tiene la idea que la mayoría de las variaciones evolutivas son neutrales.
- ◆ Teoría de la Evolución Neutral (Motoo Kimura) sugiere que diferentes formas del mismo gen son indistinguibles en sus efectos. Esto es, una mutación de un gen a otro es neutra si esta modificación no afecta el fenotipo
- ◆ No contradice a la teoría Darwiniana
- ◆ La teoría Darwiniana juzga a los individuos por su fenotipo (fitness) mientras que la teoría de Kimura argumenta que las mutaciones ocurridas dentro del proceso evolutivo no son ni ventajosas ni desventajosas para la supervivencia y reproducción de los individuos

Código Genético

- ◆ En la molécula conocida como ácido desoxiribonucleico (ADN) se encuentra codificada la información suficiente para sintetizar las proteínas necesarias para la vida.
- ◆ A pesar de la variedad de funciones que realizan las proteínas, las unidades que la constituyen son únicamente 20 (aminoácidos)
- ◆ Todos los organismos emplean un alfabeto de 4 letras para codificar los 20 aminoácidos
- ◆ Las cuatro letras del alfabeto de aminoácidos son los nucleótidos Adenina (A), Guanina (G), Citosina (C) y Timina (T)
- ◆ Los nucleótidos son las moléculas que forman la cadena de ADN

Código Genético

- ◆ Con una secuencia de longitud 3, se representan hasta $4^3 = 64$ instancias
- ◆ Solo existen 20 aminoácidos, diversas secuencias codifican el mismo aminoácido
- ◆ La secuencia de tres espacios que se asocia a un aminoácido se conoce como triplete o codón
- ◆ Un mismo aminoácido puede representarse por más de un triplete
- ◆ La Valina se relaciona con los tripletes: GTT, GTC, GTA y GTC.
- ◆ Una mutación en la tercera unidad nos llevaría al mismo aminoácidos

Código Genético

	T	C	A	G	
T	Phe(F)	Ser(S)	Tyr(Y)	Cys(C)	T
T	Phe(F)	Ser(S)	Tyr(Y)	Cys(C)	C
T	Leu(L)	Ser(S)	Ter(.)	Ter(.)	A
T	Leu(L)	Ser(S)	Ter(.)	Trp(W)	G
C	Leu(L)	Pro(P)	His(H)	Arg(R)	T
C	Leu(L)	Pro(P)	His(H)	Arg(R)	C
C	Leu(L)	Pro(P)	Gln(Q)	Arg(R)	A
C	Leu(L)	Pro(P)	Gln(Q)	Arg(R)	G
A	Ile(I)	Thr(T)	Asn(N)	Ser(S)	T
A	Ile(I)	Thr(T)	Asn(N)	Ser(S)	C
A	Ile(I)	Thr(T)	Lys(K)	Arg(R)	A
A	Met(M)	Thr(T)	Lys(K)	Arg(R)	G
G	Val(V)	Ala(A)	Asp(D)	Gly(G)	T
G	Val(V)	Ala(A)	Asp(D)	Gly(G)	C
G	Val(V)	Ala(A)	Glu(E)	Gly(G)	A
G	Val(V)	Ala(A)	Glu(E)	Gly(G)	G

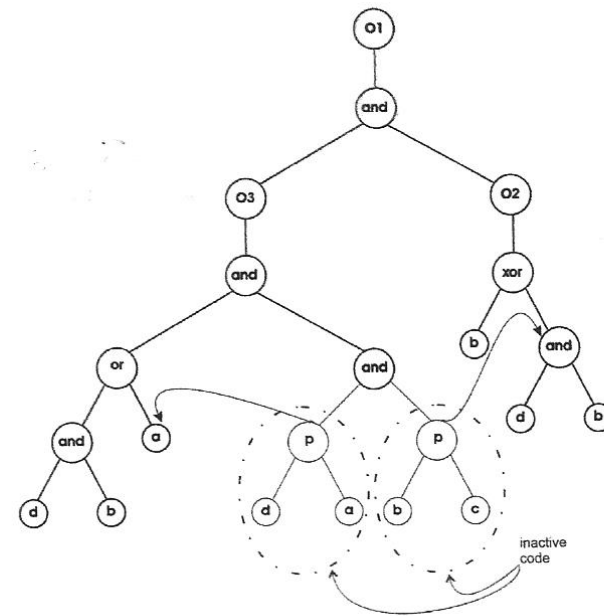
Teoría de la Evolución Neutral

- ◆ En PG, neutralidad es frecuentemente asociada con redundancia e intrones
- ◆ Redundancia funcional se presenta cuando se tienen diferentes individuos los cuales a nivel fenotipo representan la misma function
- ◆ E.j. (NOR (AND (NOT (NOT A)) B) (NOT (OR A B)))
(NOR ((NAND (NAND A B) (OR A B)) (NOT (OR A B))))
- ◆ Intrones es código parte del individuo que no se traduce al fenotipo (no afecta el comportamiento del programa)
- ◆ El problema aquí es que redundancia e intrones emergen durante el proceso evolutivo y por tanto es difícil de cuantificar

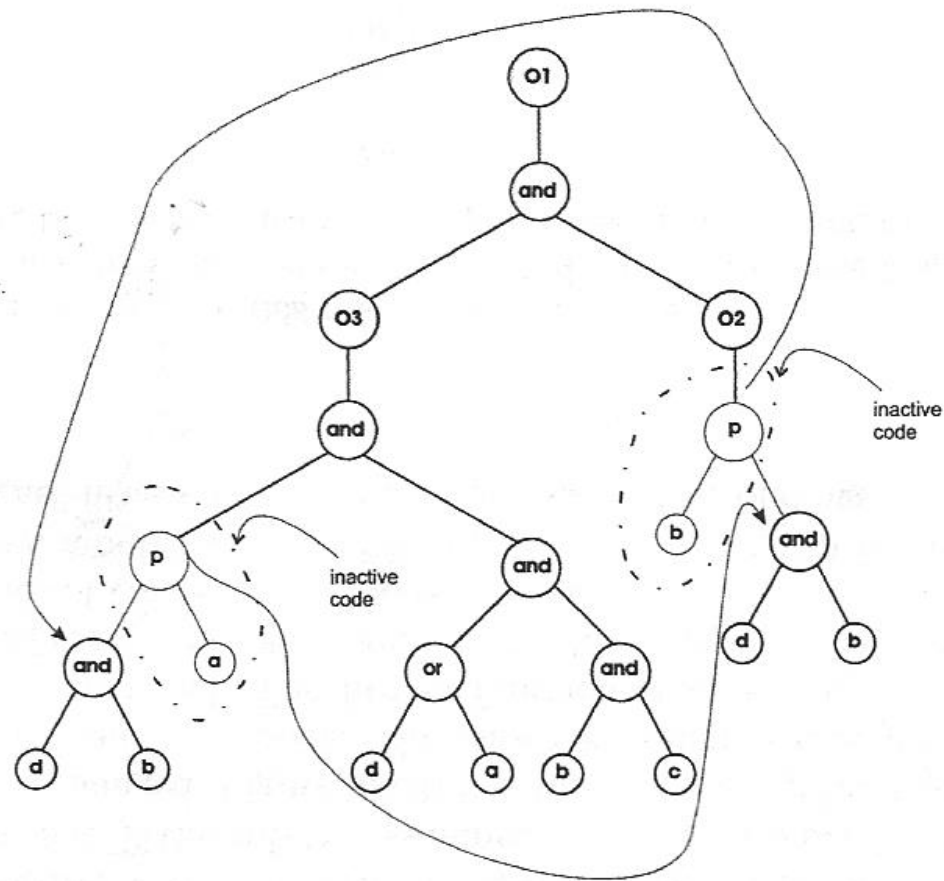
Teoría de la Evolución Neutral

- ◆ La idea es tener neutralidad explícita (nodos inactivos)
- ◆ Para este efecto se tienen diferentes tipos de representaciones

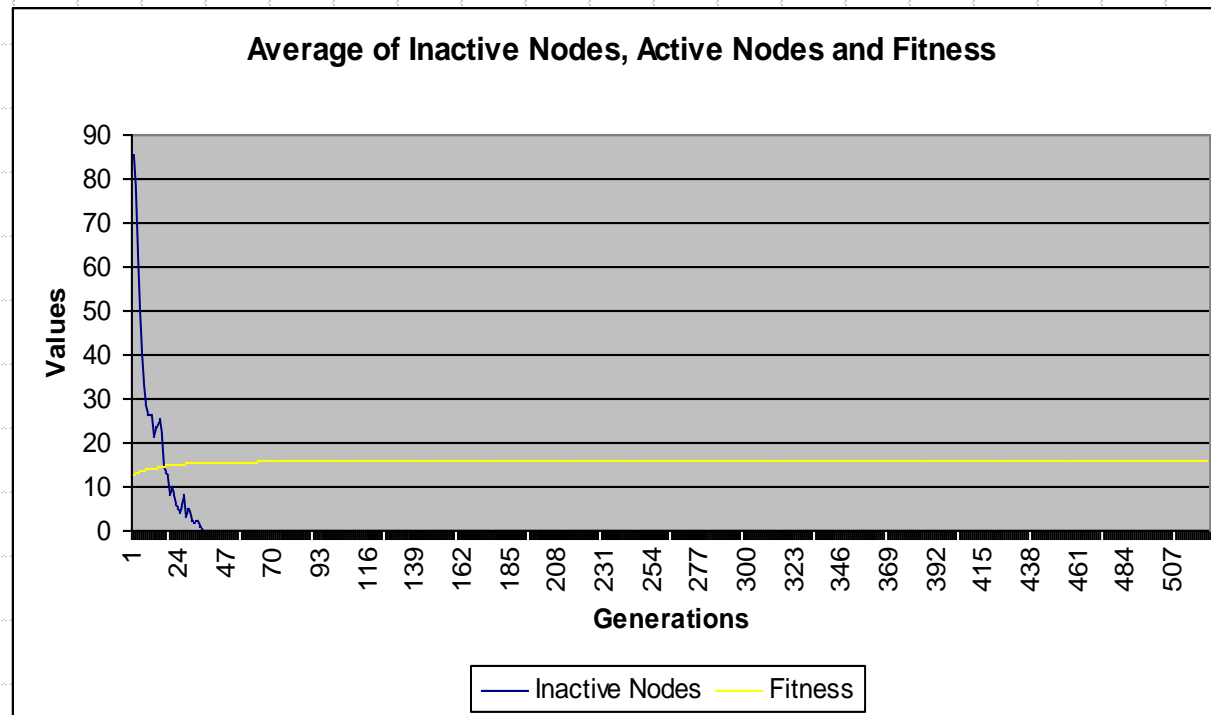
4



Teoría de la Evolución Neutral

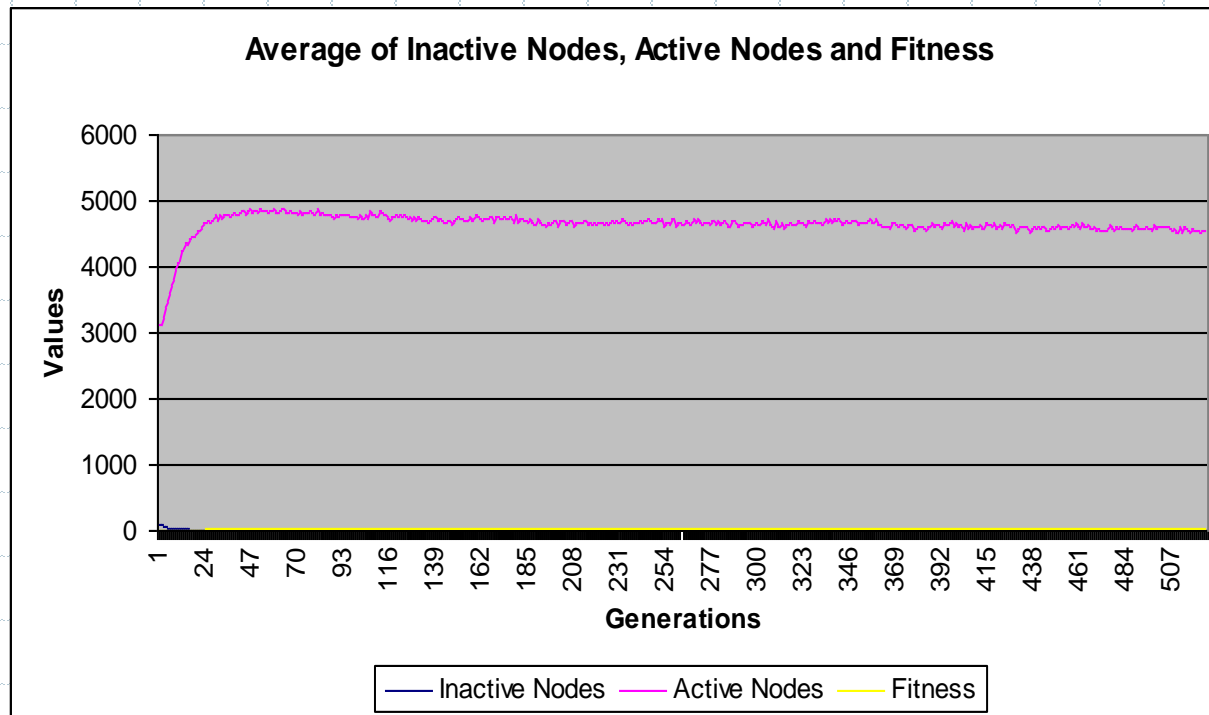


Neutralidad



Nota: Considerando neutralidad, PG resuelve el 100% de las corridas (solución funcional).

Neutralidad

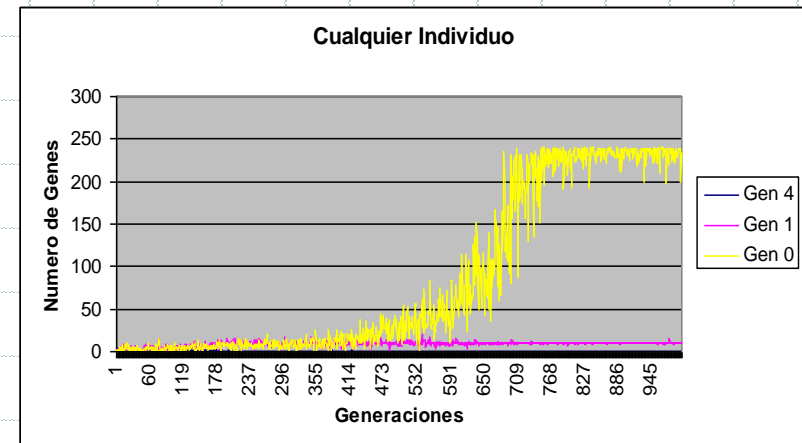


Neutralidad

- ◆ Tendencia: Presencia alta en el número de nodos inactivos en etapa temprana del proceso evolutivo antes de alcanzar zona factible, después la presencia de nodos inactivos decrece
- ◆ Por tanto, la neutralidad favorece el proceso evolutivo para alcanzar zona factible y no permite quedarse estancado en un óptimo local

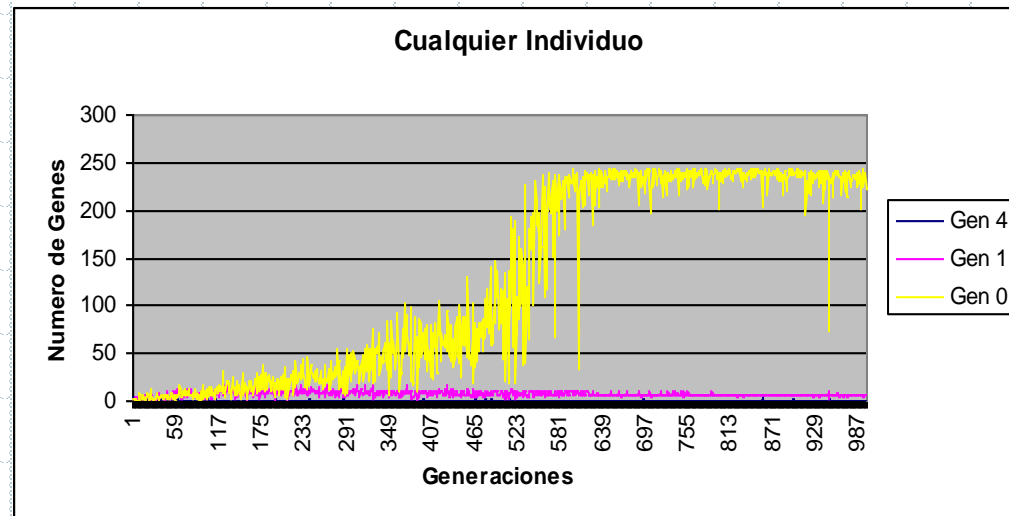
Neutralidad

Objective	Find integers whose sum is T
Integer Values	0, 1, 4
Population Size	500
Crossover	0,9
Mutation	0.01 per node
Selection	2 member tournament
Generations	1000
Maximum Size	500
Elitism	2 copies of the best individuals are preserved
Initial Population	random individual of length 9 to 128
Number of trials	200
Crossover	Random points
Mutation	Replace per node



Neutralidad

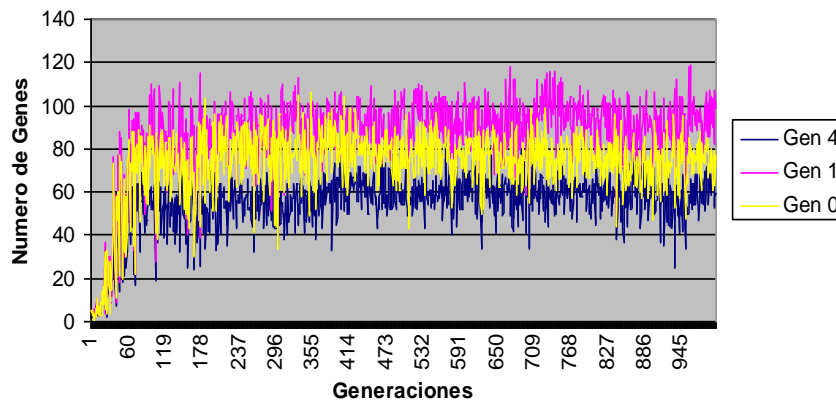
Objective	Find integers whose sum is T
Integer Values	0, 1, 4
Population Size	500
Crossover	0,9
Mutation	0,1
Selection	2 member tournament
Generations	1000
Maximum Size	500
Elitism	2 copies of the best individuals are preserved
Initial Population	random individual of length 9 to 128
Number of trials	200
Crossover	Random points
Mutation	Replace a old branche for a new one



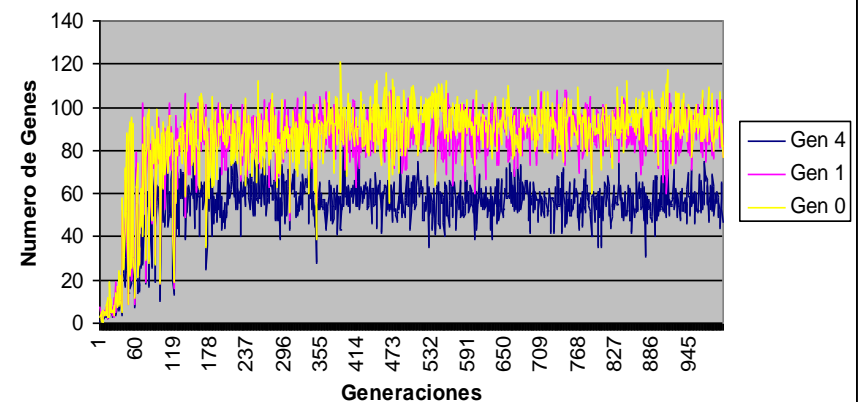
Neutralidad

Corrida 0

Mejor Individuo Sin P's



Mejor Individuo - P's



Objective	Find individual whose value is T
Integer Values	0, 1, 4
Population Size	500
Crossover	0,9
Mutation	0,1
Selection	2 member tournament
Generations	1000
Maximum Size	500
Elitism	2 copies of the best individuals are preserved
Initial Population	random individual of length 9 to 128
Number of trials	200
Crossover	Random points
Mutation	Replace a old branche for a new one

Function Set = { '+', '*' }

Neutralidad

- ◆ Otras representaciones con nodos inactivos (representación de multi-ramas):

```
(ADD
V1
(divd (. * V4 (. * V4 V4)) V5)
(divd (divd V1 (divd V1 V4)) V5)
(. * V1 V4)
(divd (divd (. * V2 V2) (. * V5 V2)) V5)
(divd V3 (divd V5 V1))
(divd V3 V5)
(. * (+ V1 (divd V4 V5)) V4)
(divd V3 (divd V5 (- V3 V1)))
3.3067 3.5036 0.87467 0.82177 0.0060857 0.70499
-0.2266 -2.6548 0.59913 -1.6498
)
```

Neutralidad

```
(OR
  0
  0
  0
  (xor (xor (not (xor V1 V4)) V2)
    V3)
)
```

```
(OR
  (xor V2 (xor V3 (xor V4 (not
    V1))))
  (xor V2 (xor V3 (xor V4 (not
    V1))))
  (xor V2 (xor V3 (xor V4 (not
    V1))))
  (xor V2 (xor V3 (xor V4 (not
    V1))))
)
```

```
(OR
  0
  0
  0
  (not (xor (xor V2 (xor V4 V3)) V1))
)
```

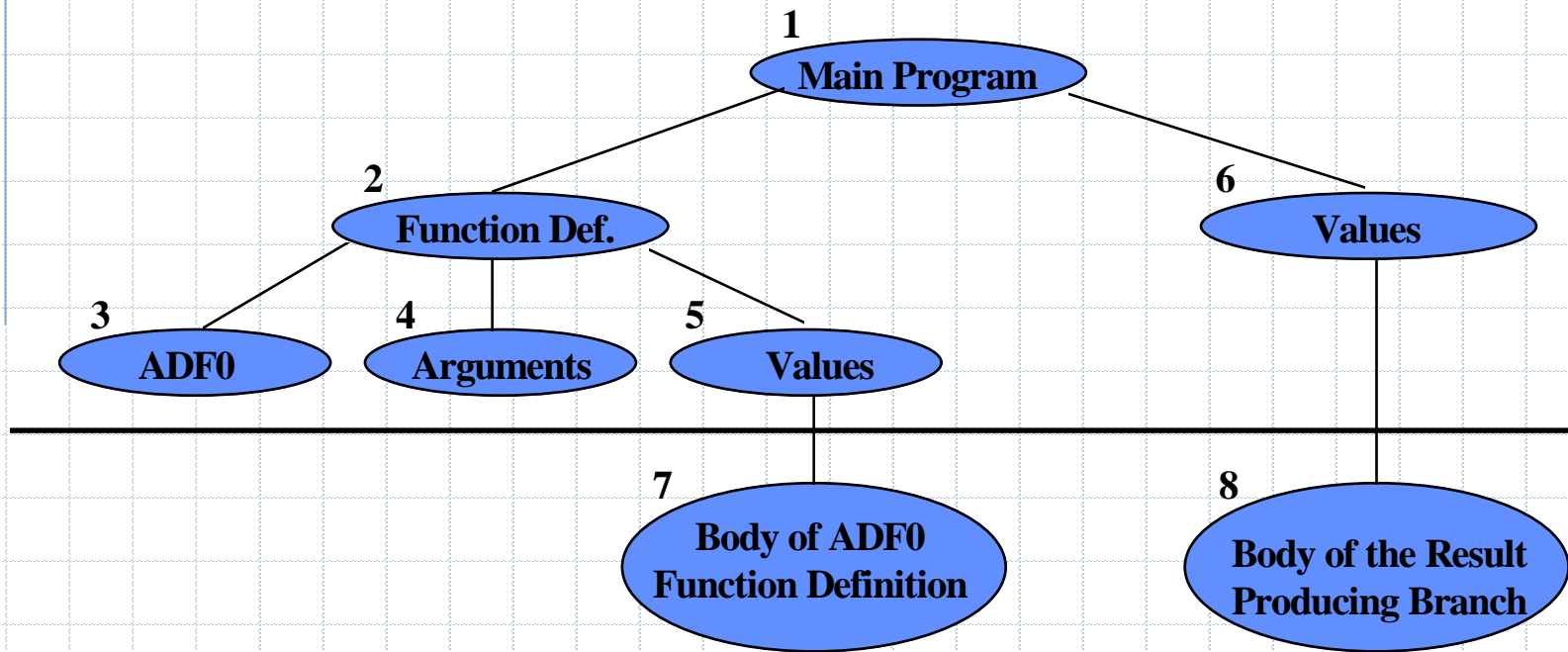
ADF's

□ ***Funciones Definidas Automaticamente (Automatically Defined Functions, ADF's)*** [Koza, 1994]

Función (e.j. sub-rutina, procedimiento, módulo) la cual evoluciona durante una ejecución de GP y la cual puede ser llamada por un programa (e.j. programa principal) el cual está simultáneamente evolucionando.

GP + ADF mantienen un patrón fijo durante todo el proceso de búsqueda, el cual contiene dos secciones que puede variar (una result-producing branch [programa principal] y una o más function-defining branch [ADF's])

ADF's



ADF's

Problema de las dos cajas

Encontrar un programa el cual calcule la diferencia en volumen entre dos cajas.

$$F = \{+, -, *, \%, \text{ADF0}\}$$

$$T = \{H_0, L_0, W_0, H_1, L_1, W_1\}$$

Solución en notación polaca:

$$(- \text{ (VOLUME (L}_0 \text{ W}_0 \text{ H}_0)) \text{ (VOLUME (L}_1 \text{ W}_1 \text{ H}_1)))$$

equivalente:

$$L_0 W_0 H_0 - L_1 W_1 H_1$$

ADF's

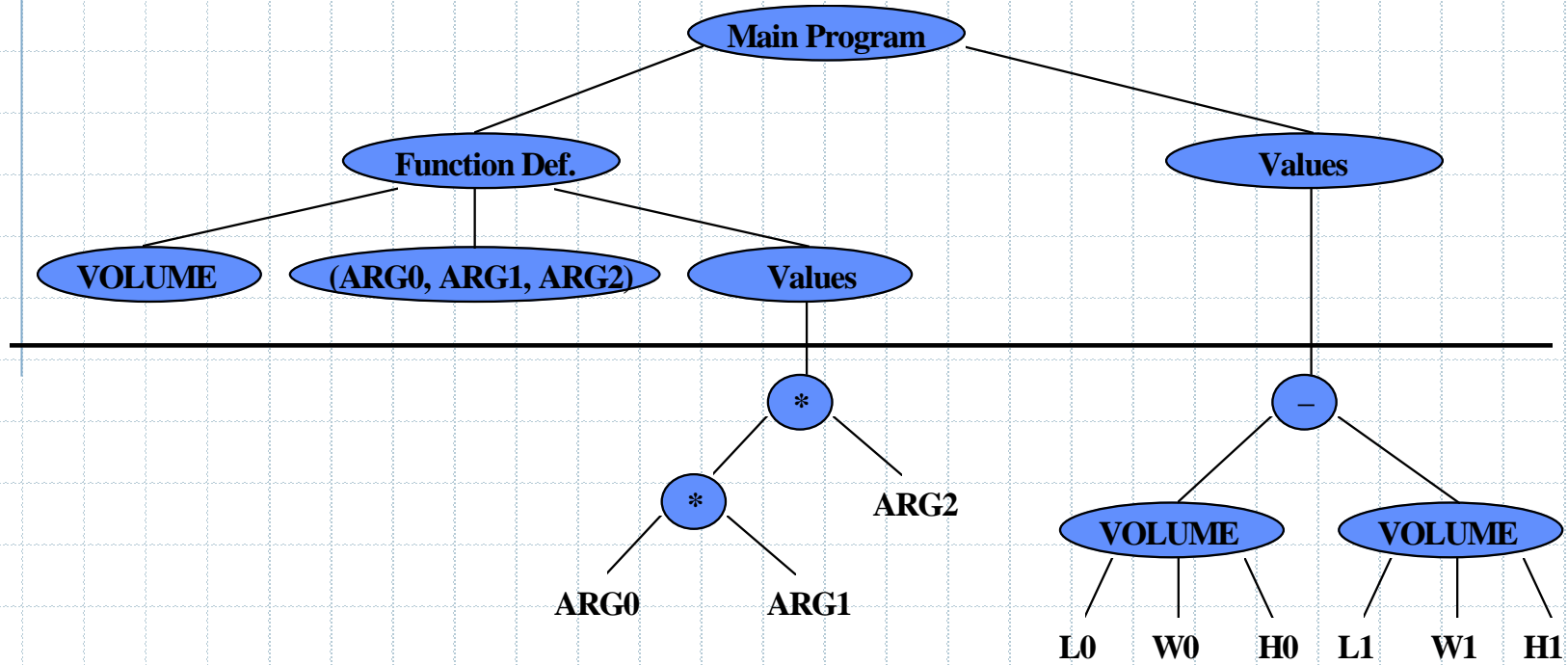
$$F = \{+, -, *, \%, \text{ADF0}\}$$

$$T = \{H_0, L_0, W_0, H_1, L_1, W_1\}$$

$$F_{\text{ADF0}} = \{+, *, \sin, \cos\}$$

$$T_{\text{ADF0}} = \{\text{ARG}_0, \text{ARG}_1, \text{ARG}_2\}$$

ADF's

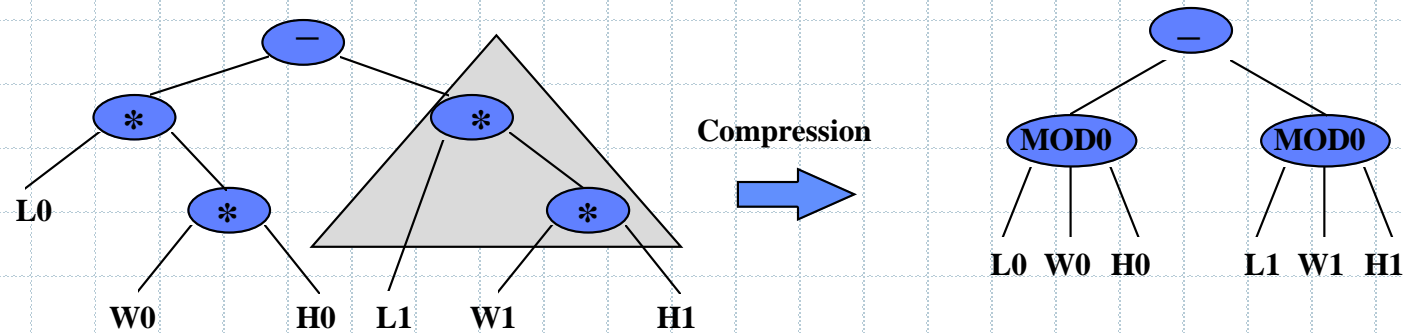


Genetic Library Builder (GLiB) and Module Acquisition (MA)

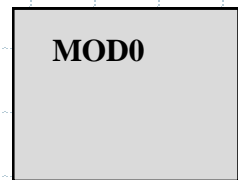
- Angeline and Pollack (1993)
- Crea librerías (Module Acquisition, MA) de módulos únicos, globalmente definidos extendiendo el conjunto de funciones
- GLiB considera dos nuevos operadores: **compresión** y **expansión**
- **Compresión:** selecciona un sub-árbol de un individuo padre
módulo nuevo en el conjunto de funciones
El sub-árbol en el padre es reemplazado por la función MOD0
Módulos no evolucionan
- **Expansión:** reemplaza un módulo por su definición almacenada en las librerías genéticas

Genetic Library Builder (GLiB) and Module Acquisition (MA)

Module Acquisition (MA)



Module Library



$\mathcal{F} = \{+, -, *, \%, \log, \exp, \sin, \dots, \text{MOD0}\}$

$\mathcal{T} = \{L0, W0, H0, L1, W1, H1\}$

