Computación Evolutiva II

Jose Aguilar
Cemisid, Facultad de Ingeniería
Universidad de los Andes
Mérida, Venezuela
aguilar@ula.ve

PROGRAMACIÓN GENÉTICA

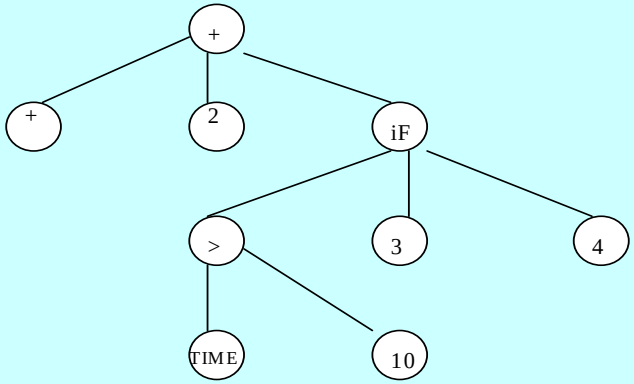
Consiste en utilizar la metodología de CE no para encontrar soluciones a problemas, sino para encontrar el mejor procedimiento para resolverlos.

Las estructuras que se someten a evolución codifican los algoritmos o programas que, al ser ejecutados, determinan soluciones.

PROGRAMACIÓN GENÉTICA

La representación de los individuos se hace a través de *árboles de análisis.*

En la practica, se usa el lenguaje LISP: Por ejemplo, la siguiente expresión: *(+ 1 2 (IF (> TIME 10) 3 4))*, su árbol es:



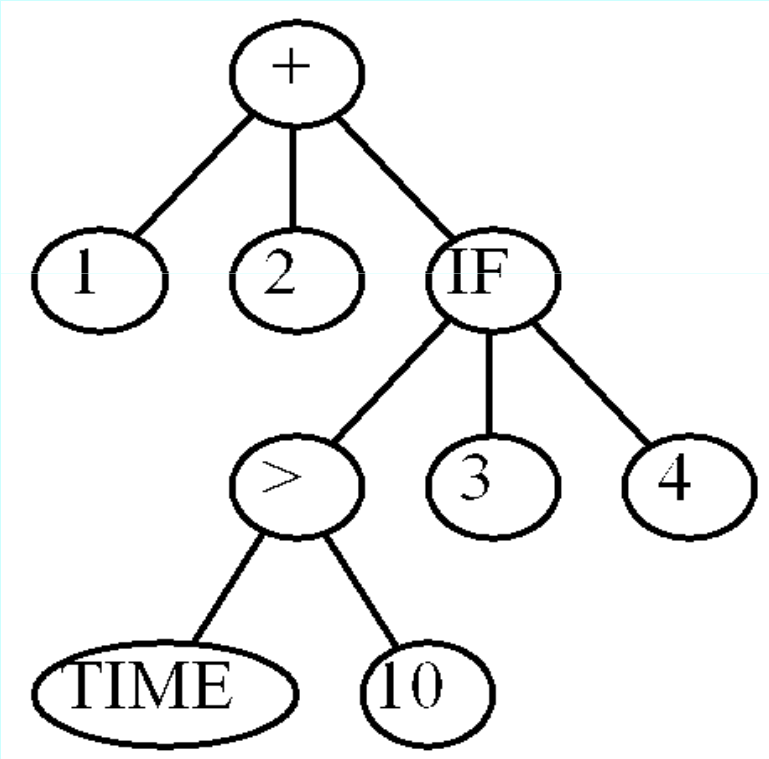
PROGRAMACIÓN GENÉTICA

```
int foo (int time)
{
    int temp1, temp2;
    if (time > 10)
        temp1 = 3;
    else
        temp1 = 4;
    temp2 = temp1 + 1 + 2;
    return (temp2);
}
```

Time	Output
0	6
1	6
2	6
3	6
4	6
5	6
6	6
7	6
8	6
9	6
10	6
11	7
12	7

PROGRAMACIÓN GENÉTICA

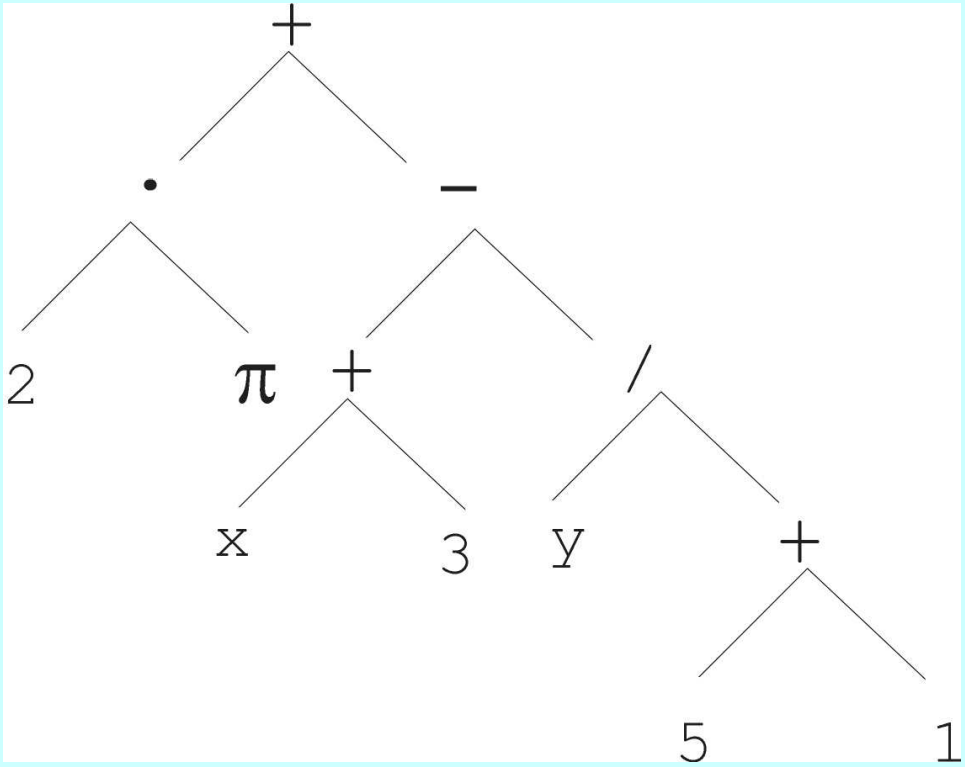
Representación de un programa




```
(+ 1 2 (IF (> TIME 10) 3 4))
```

PROGRAMACIÓN GENÉTICA

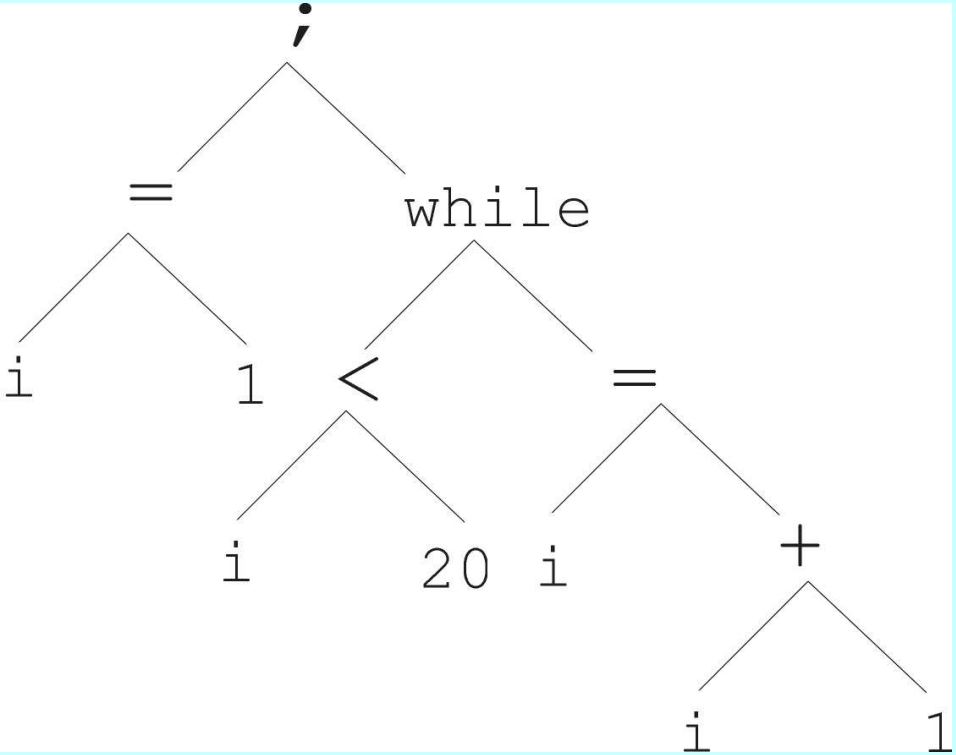
Representaciones de una fórmula



$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

PROGRAMACIÓN GENÉTICA

Representaciones basadas en árboles




```
i =1;
while (i < 20)
{
        i = i +1
}
```

PROGRAMACIÓN GENÉTICA

La codificación utiliza un conjunto de *símbolos de funciones* y otro de *símbolos terminales* (átomos) adecuados para la solución del problema dado.

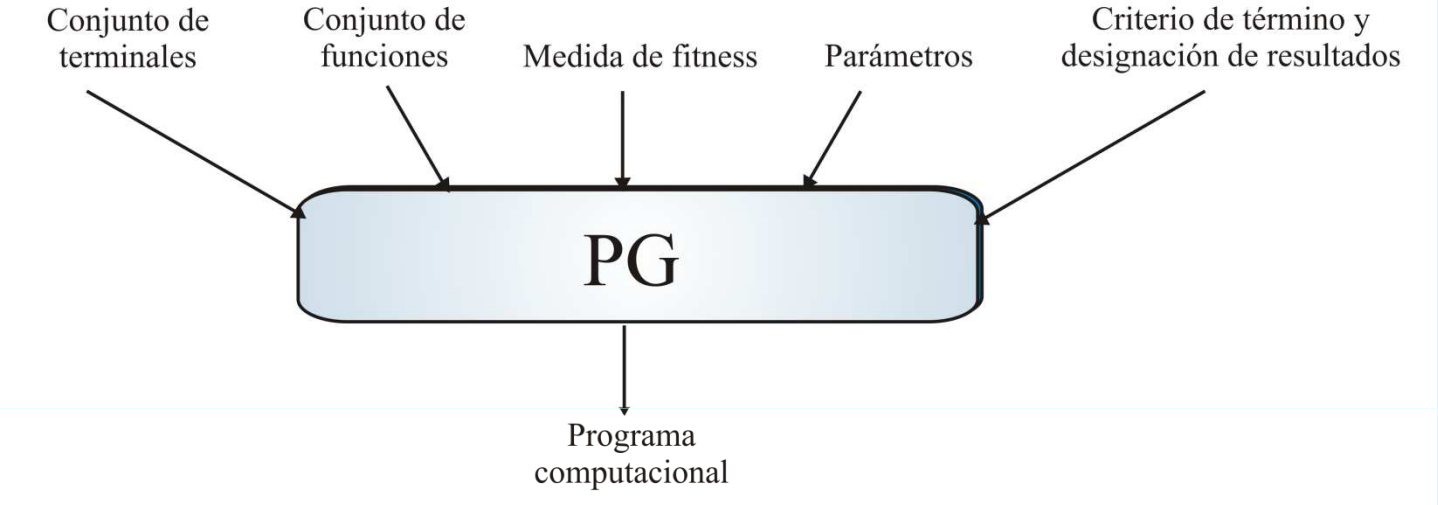
Las funciones pueden ser operaciones aritméticas, funciones matemáticas, funciones lógicas, o funciones especificas del dominio.

PROGRAMACIÓN GENÉTICA

ELEMENTOS:

- CONJUNTO DE TERMINOS
- CONJUNTO DE FUNCIONES
- MEDIDA DE APTITUD
- PARAMETROS DE CONTROL
- CRITERIO DE CULMINACION

PROGRAMACIÓN GENÉTICA



- Determinar el conjunto de terminales
- Determinar el conjunto de funciones
- Determinar la medición del fitness
- Determinar los parámetros para la ejecución
- Determinar el método para designar un resultado y un criterio para terminar una ejecución

Conjuntos de Funciones y Terminales

Los nodos en el árbol pueden ser:

- Conjunto de Funciones F
  - El conjunto de todos los posibles nodos internos en el árbol
  - Todas las funciones tienen una ariedad ≥ 1.
- Conjunto de terminales T
  - El conjunto de todos los posibles nodos hoja en el árbol

El espacio de búsqueda del problema son todas las posibles combinaciones de funciones y terminales

Parámetros de control

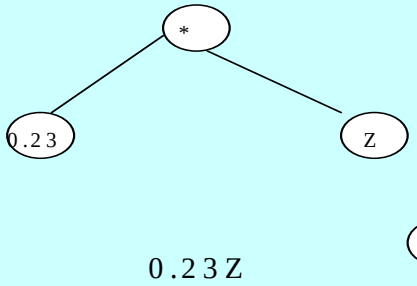
- Los parámetros de control más importantes son:
  - **G:** número máximo de generaciones a producir
  - **M:** tamaño de la población
  - **P_C:** probabilidad de cruzamiento
  - **P_M:** probabilidad de mutación
  - **D_{max}** tamaño máximo (medido por profundidad) de los individuos creados por las operaciones de evolución darwiniana
  - **$D_{inicial}$** tamaño (medido por profundidad) de los individuos creados en la población inicial

PROGRAMACIÓN GENÉTICA

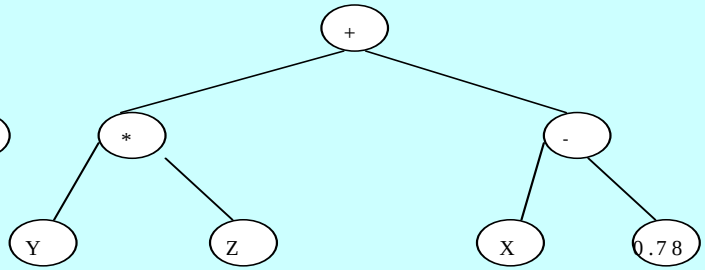
EJEMPLO:

CONJ. TERMINOS:{X,Y,Z, REALES}

CONJ. FUNCIONES:{+,-,*,IF}

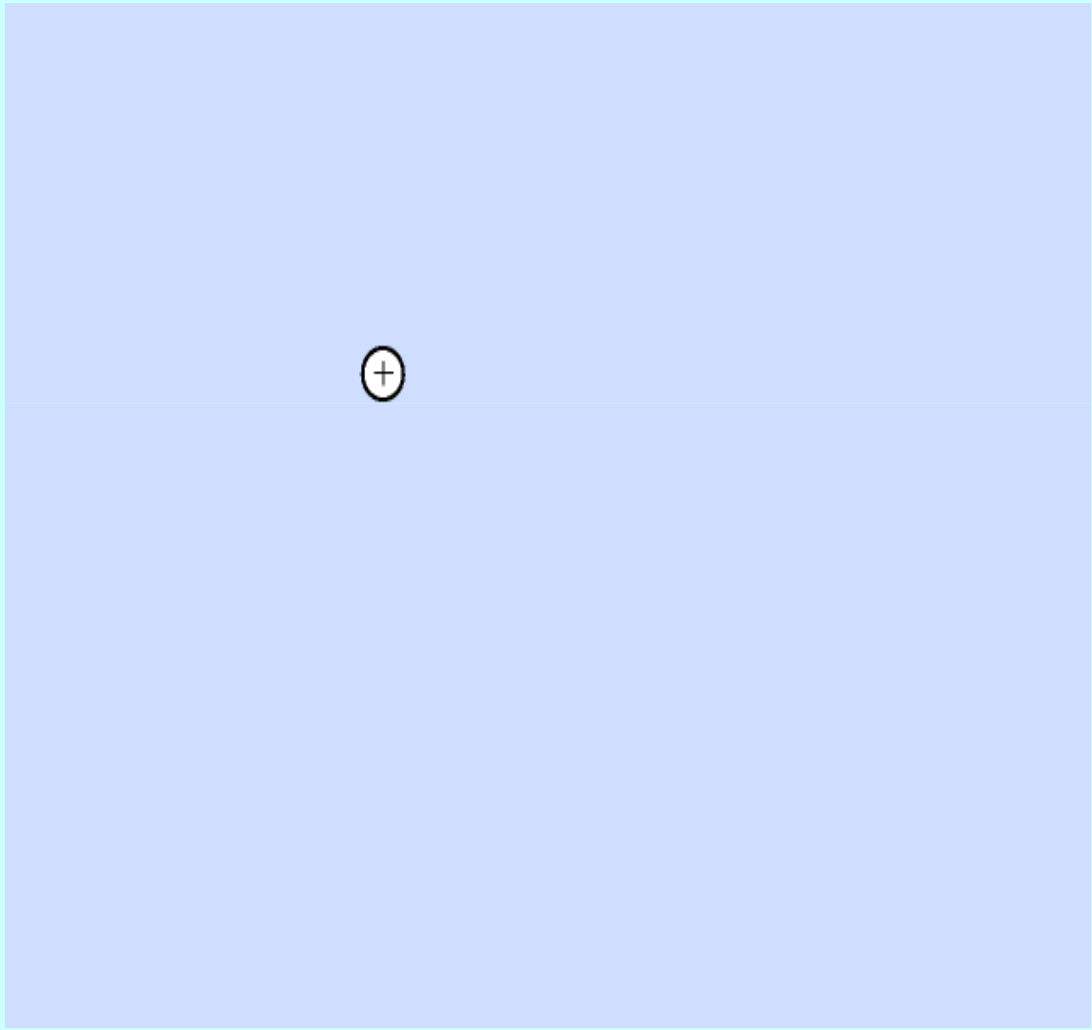


0.23Z



Y Z + X -0.78

Creando programas aleatorios



La función de Evaluación

- Función de Fitness “de Error”

$$f_p = \sum_{i=1}^{n} |p_i - o_i|$$

p : Algoritmo
p_i : la salida del programa p en el i^{ésimo} caso de fitness
o_i : la salida del i^{ésimo} ejemplo en el caso de fitness
f_p : el fitness de p.
n : número de casos de fitness

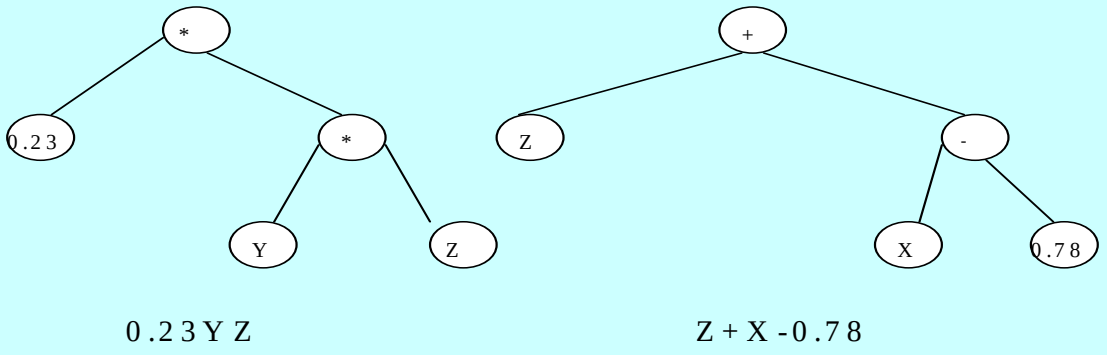
– Función de evaluación de error cuadrático $$f_p = \sum_{i=1}^{n} (p_i - o_i)^2$$

PROGRAMACIÓN GENÉTICA

OPERADOR CRUCE

0.23Z YZ+X-0.78

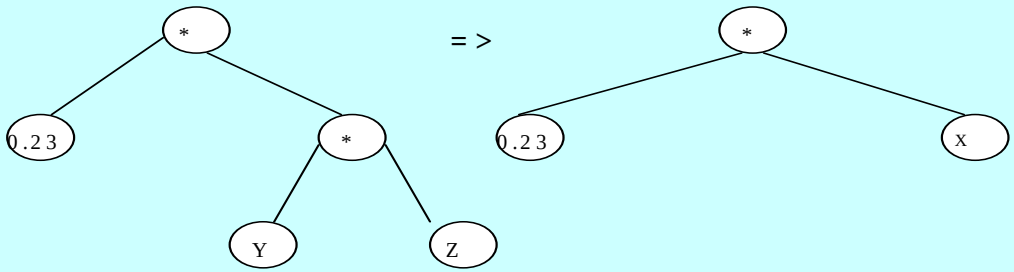
=> 0.23YZ Z+X-0.78



PROGRAMACIÓN GENÉTICA

OPERADOR MUTACION

0.23YZ

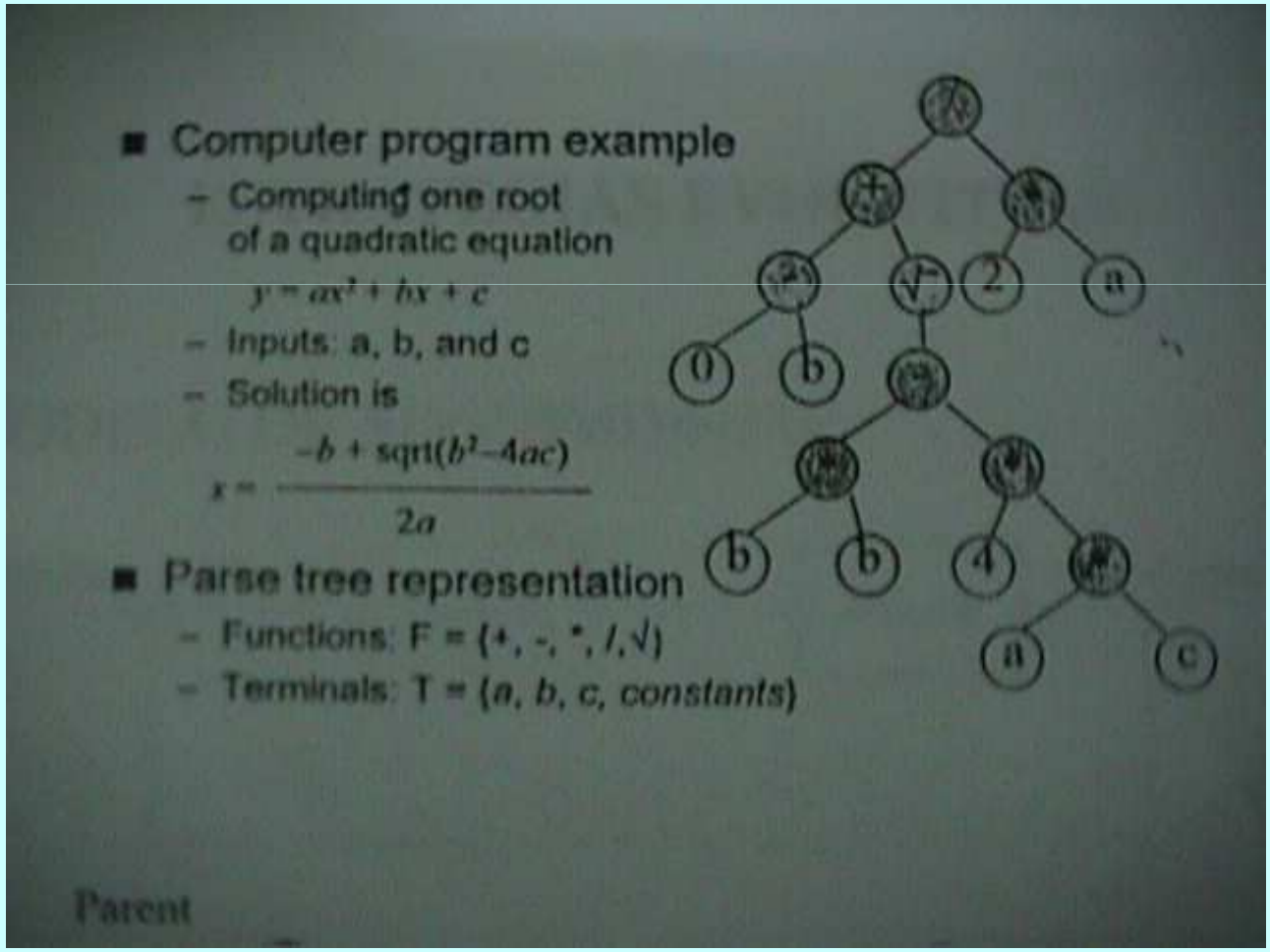


0.23YZ 0.23X

PROGRAMACIÓN GENÉTICA

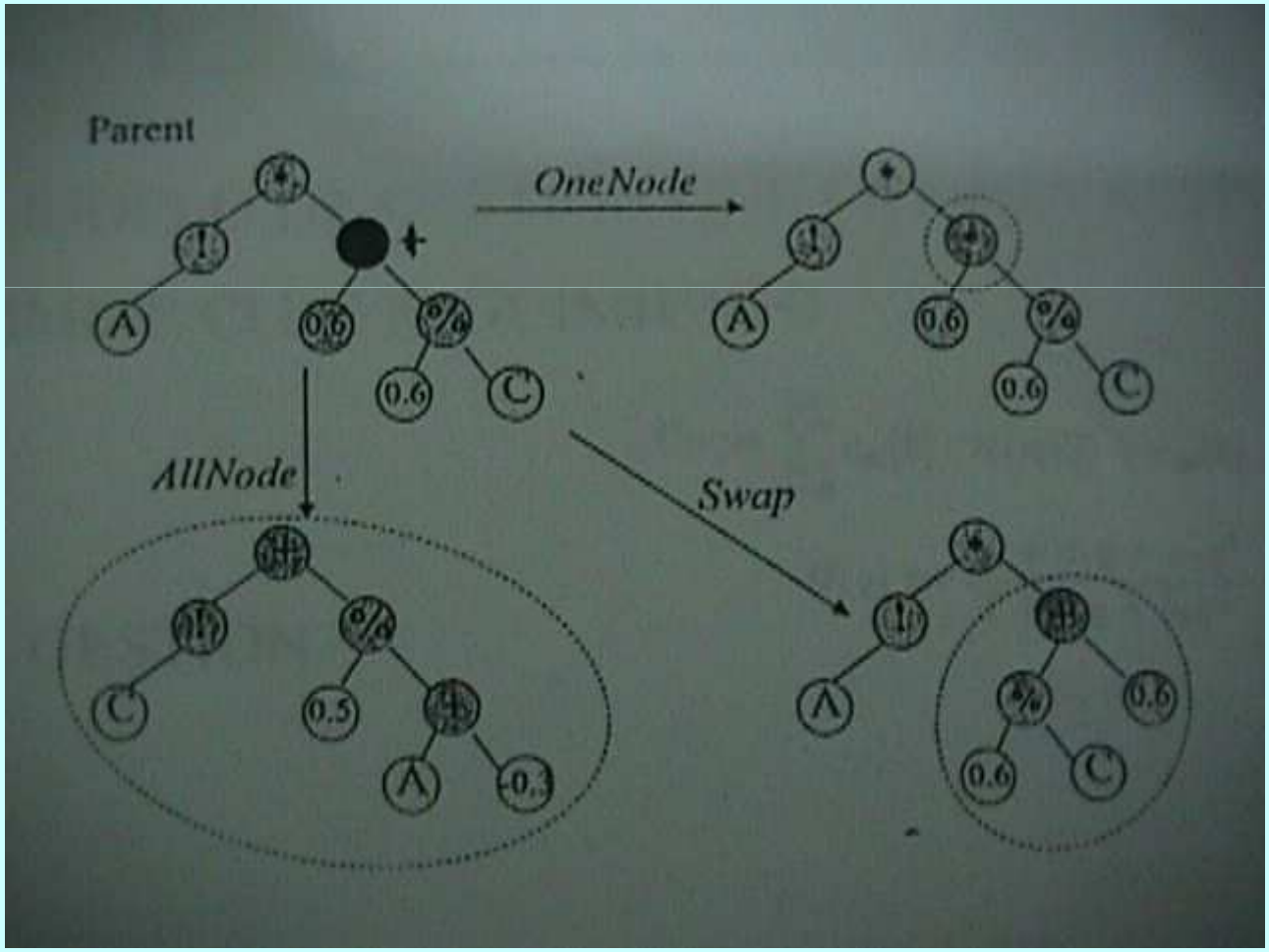
EJEMPLO PARA PRESENTAR OTROS OPERADORES

SUPONGA



PROGRAMACIÓN GENÉTICA

OTROS OPERADORES



PROGRAMACIÓN GENÉTICA

EJEMPLO: PROBLEMA DE REGRESION DE FUNCION DESCONOCIDA

VAR. IND: X VAR DEP: Y

X	Y
-1	0.178
-0.9	-0.169
...	
0	0.00
...	

PROGRAMACIÓN GENÉTICA

PROBLEMA: ENCONTRAR Y=F(X)

TERMINOS:X

FUNCIONES: +,-,*,SIN COS, EXP, LOG

FUNC. ADAPTAT.: ERROR

GENER 0:	X+ LOG(2X)+X	fa: 6.05
	COS(COS(+(-(*XX))))	fa: 23,67
GENER 34:	$X^4+X^3+X^2+X$	fa: 0

PROGRAMACIÓN GENÉTICA

Problema de la mochila

$$Max\ Z = \sum_{j=1}^{n} p_j x_j$$

Sujeto a

$$\sum_{j=1}^{n} w_j x_j \leq W$$

$$x_j \in \{0,1\}, j = 1,2,3 \ldots . n$$

Código Terminal	Descripción
FALSO/VERDADERO/NULO	Valores lógicos
GT_W	Selecciona elemento de mayor Peso.
LO_W	Selecciona elemento de menor Peso.
GT_P	Selecciona elemento de mayor Beneficio.
LO_P	Selecciona elemento de menor Beneficio.
LO_R	Selecciona elemento de menor Eficiencia.
ANY	Selecciona elemento aleatorio.
Seq (subárbol P1, subárbol P2)	Operador que recibe y ejecuta dos subárboles (sub-programas) en secuencia, no retorna resultados.
While(subárbol Cond, subárbol P1)	Operador que produce una ejecución iterativa sobre el segundo subárbol, según se cumpla la condición establecida en el primer subárbol. Se maneja la condición con dos variables extras, las cuales son, número de veces que se ejecutó el ciclo "while" sin tener un efecto real en la configuración de la instancia, denominado ITER_SIN_EFECTO. Este valor se fijó en *n* (número de objetos de la mochila) y número máximo de iteraciones de ciclo "while", denominado ITER_MAX. Este valor fue fijado en 2*n.
Put (subárbol P1)	Agrega un elemento no seleccionado a la mochila, si el elemento no es válido o ya fue seleccionado, esta función no realiza ninguna acción.
Quitar (subárbol P1)	Remueve un elemento que ha sido seleccionado previamente
Probar (subárbol P1)	Verifica si la mochila continúa siendo válida al intentar insertar un elemento a la mochila.
SiMejora(subárbol P1)	Intercambia el objeto de menor eficiencia insertado en la mochila LO_R por P1, siempre que se cumplan las siguientes condiciones, que al intercambiar LO_R por P1, la mochila sigue estando válida o P1 es mejor que LO_R.

Problema de la mochila

Definición de la función de evaluación

Función de evaluación

$$q_p = \frac{1}{n_f} \sum_{j=1}^{n_f} \frac{u_j - z_j}{u_j}$$

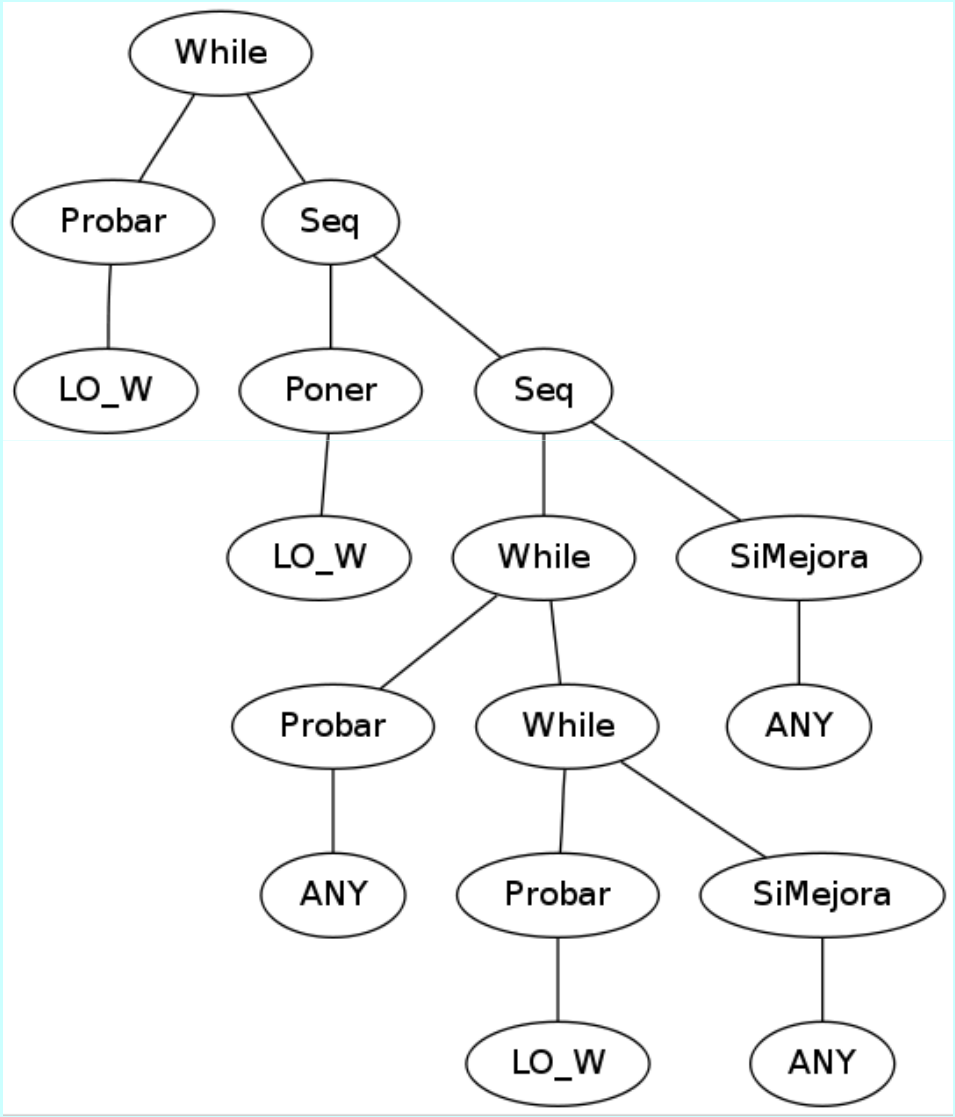
$$S_p = m / n_p$$

$$f_p = \alpha q_p + (1 - \alpha) S_p, \ \alpha \in [0,1]$$

Problema de la mochila

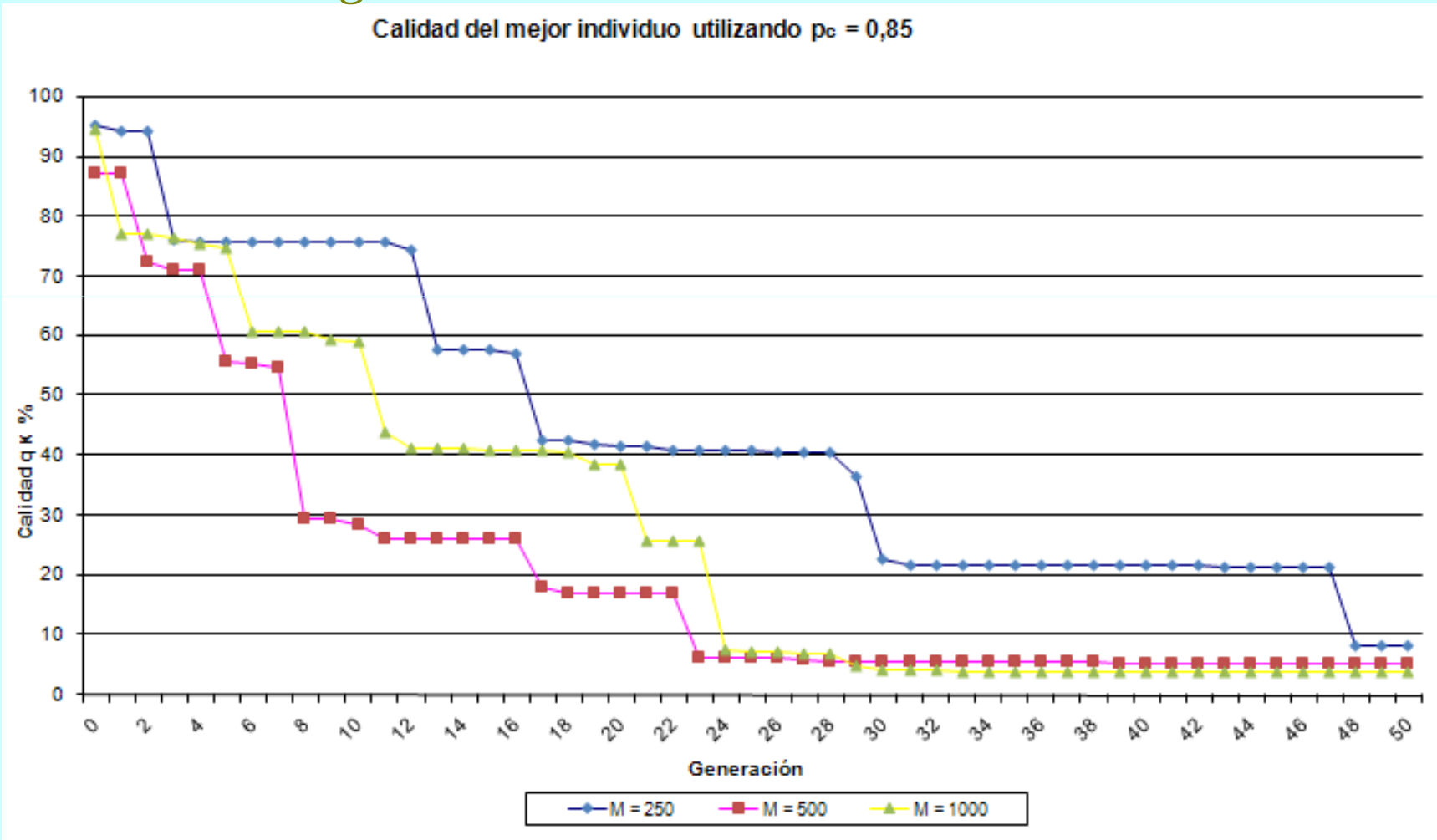
Tamaño de la población (M)	500
Numero máximo de generaciones (G)	50
Probabilidad de cruzamiento (Pc)	85%
Probabilidad de mutación (Pm)	0%

Problema de la mochila



Problema de la mochila

Convergencia



Problema de Coloración de Vértices

Dado un grafo no dirigido $G = (V, E)$, donde V es el conjunto de vértices y E el conjunto de aristas, existe una función $F: V \rightarrow N$ donde cada uno de los vértices adyacentes reciben un color distinto en N; esto es, si $u,v \in V$, entonces $F(u)!=F(v)$. El problema consiste en encontrar el menor número de colores necesarios para realizar una coloración en G.

Problema Coloración de Vértices:
Formulación matemática

$$\min \sum_{k=1}^{n} y_k$$

$$\sum_{k=1}^{n} x_{ik} = 1 \quad \forall\, i \in V$$

$$x_{ik} + x_{jk} \leq y_k \quad \forall\, (i,j) \in V, k = 1,\ldots,n$$

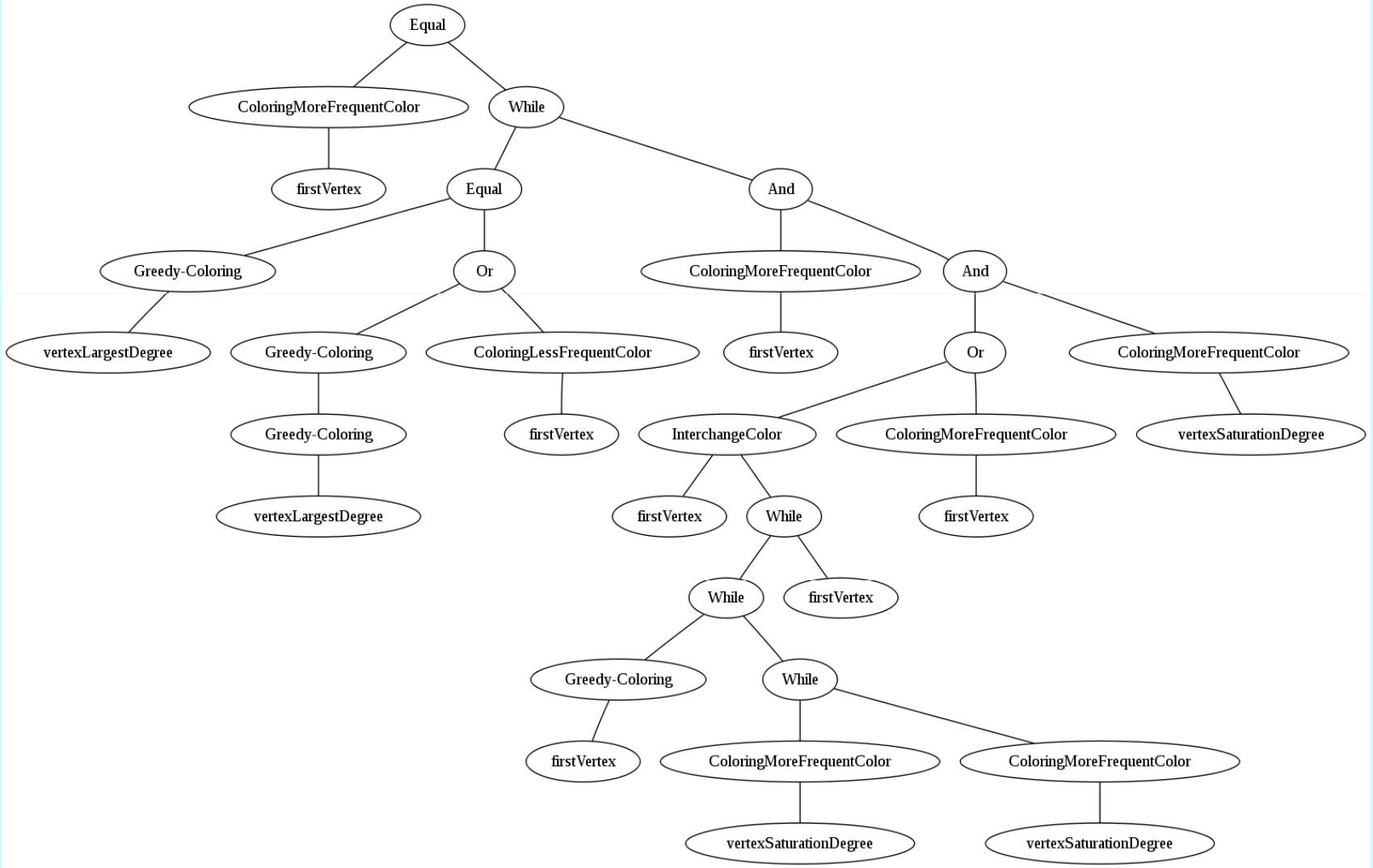
$$x_{ik} \in \{0,1\} \quad \forall\, i \in V, k = 1,\ldots,n$$

$$y_k \in \{0,1\} \quad k = 1,\ldots,n$$

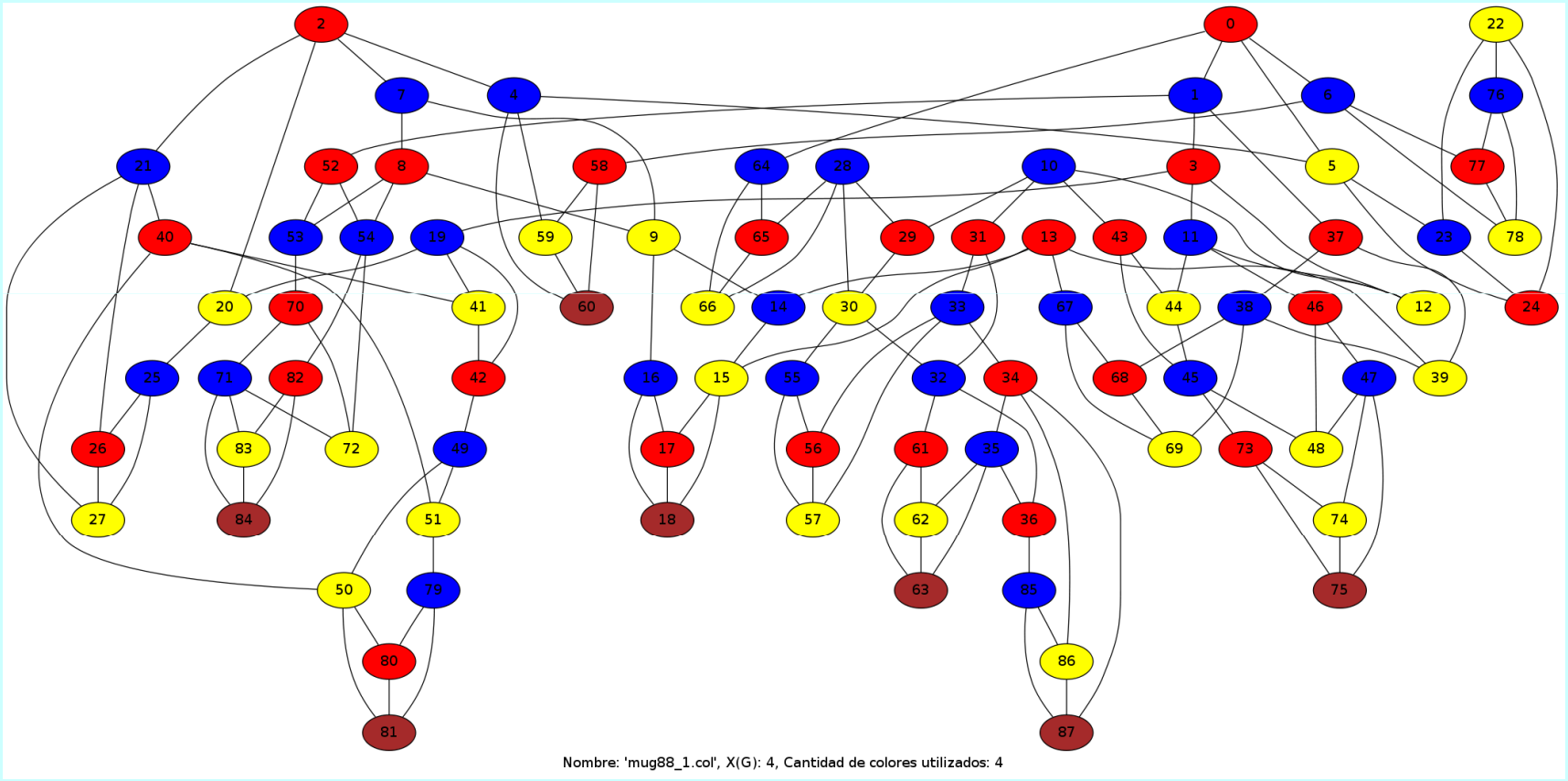
Problema Coloración de Vértices:
Parámetros

Tamaño de la población (M)	1000
Numero máximo de generaciones (G)	500
Probabilidad de cruzamiento (Pc)	80%
Probabilidad de mutación Shrink (Pm)	3%

Problema Coloración de Vértices:
Estructura algorítmica

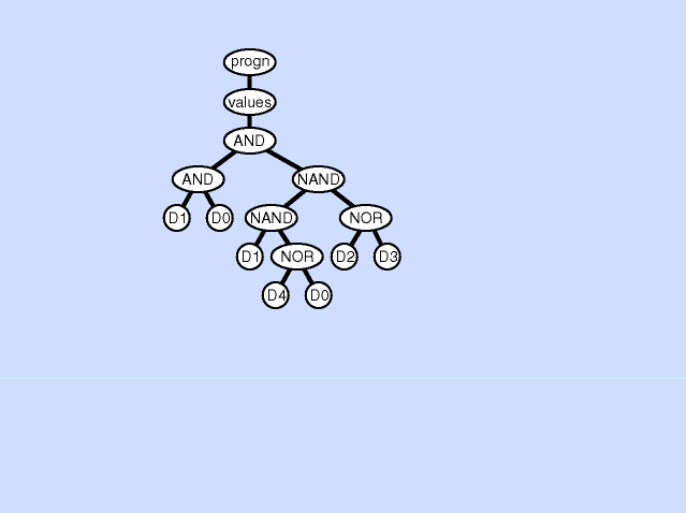


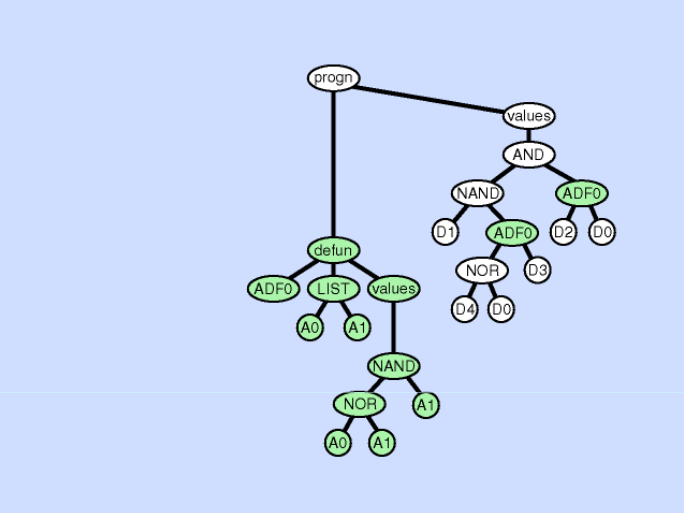
Problema Coloración de Vértices:
Algunas coloraciones

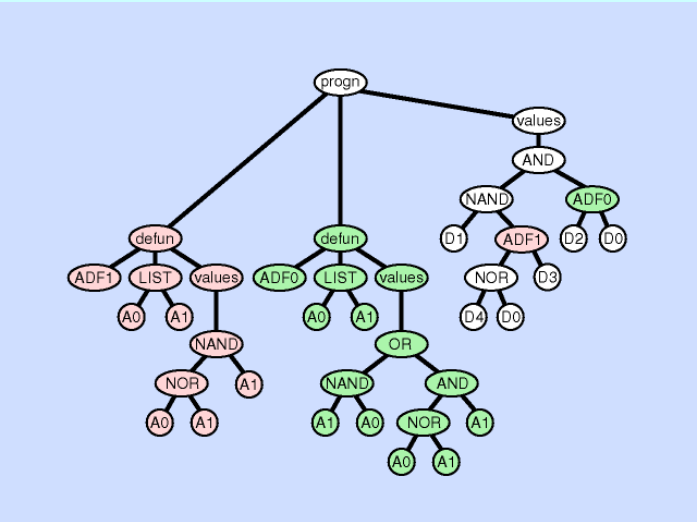


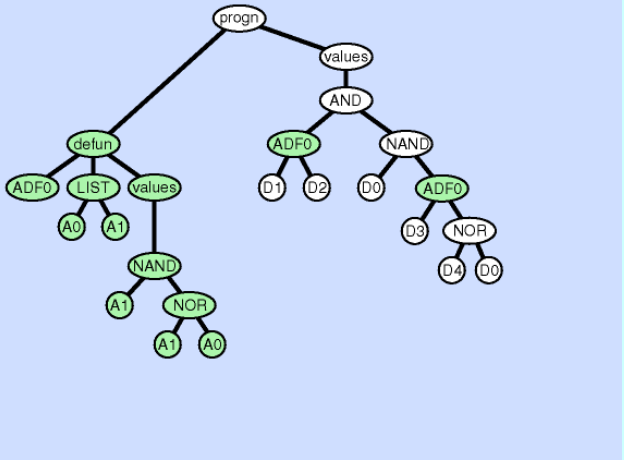
Nombre: 'mug88_1.col', X(G): 4, Cantidad de colores utilizados: 4

Operaciones que alteran la Arquitectura









PROGRAMACIÓN GENÉTICA

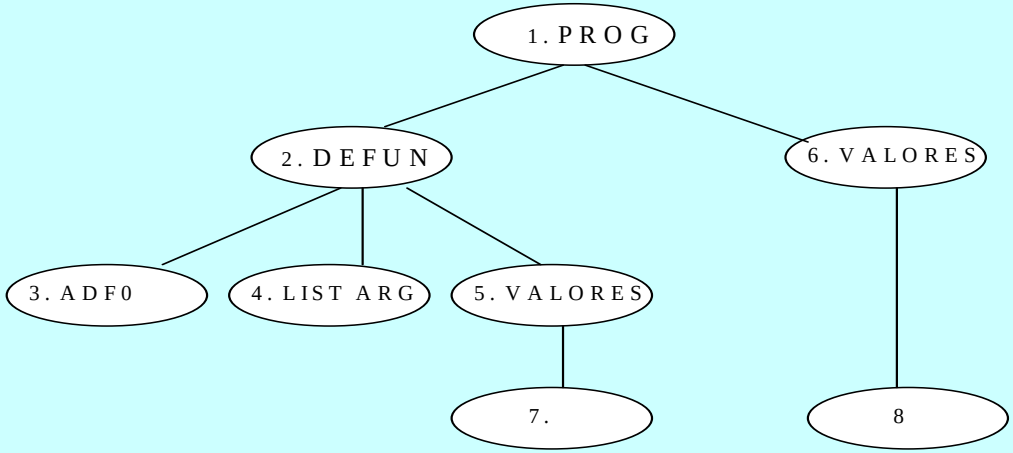
ADF (Automatically Defined Functions)

- DESCUBRIMIENTO AUTOMATICO DE UNIDADES FUNCIONALES
- INSPIRACION: HUMANOS ESCRIBEN FUNCIONES/RUTINAS QUE AGRUPAN PORCIONES DE INSTRUCCIONES QUE PUEDEN SER LLAMADAS VARIAS VECES DESDE PROGRAMAS
  - DESCOMPOSICION DE PROBLEMAS
  - RESOLUCION DE SUBPROBLEMAS
  - RECUPERACION DE LAS SOLUCIONES PARCIALES

PROGRAMACIÓN GENÉTICA

ADF (Automatically Defined Functions)

- DEFINICION DE UN ADF

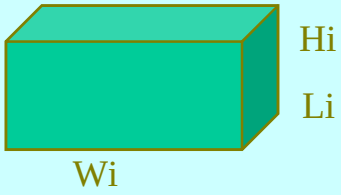


1. RAIZ
2. INICIO RAMA DEF. FUNCION
3. NOMBRE ADF
4. LISTA ARGUMENTOS
5. VALORES A REGRESAR
6. VARIAB. QUE GUARDAN RES.
7. CUERPO ADF
8. CUERPO PROGRAMA

PROGRAMACIÓN GENÉTICA

EJEMPLO ADF

- PROGRAMA CALCULA DIF ENTRE 2 CUBOS



ENFOQUE CLASICO: (-(*(* W0 L0)) H0) (* (* W1 L1) H1))

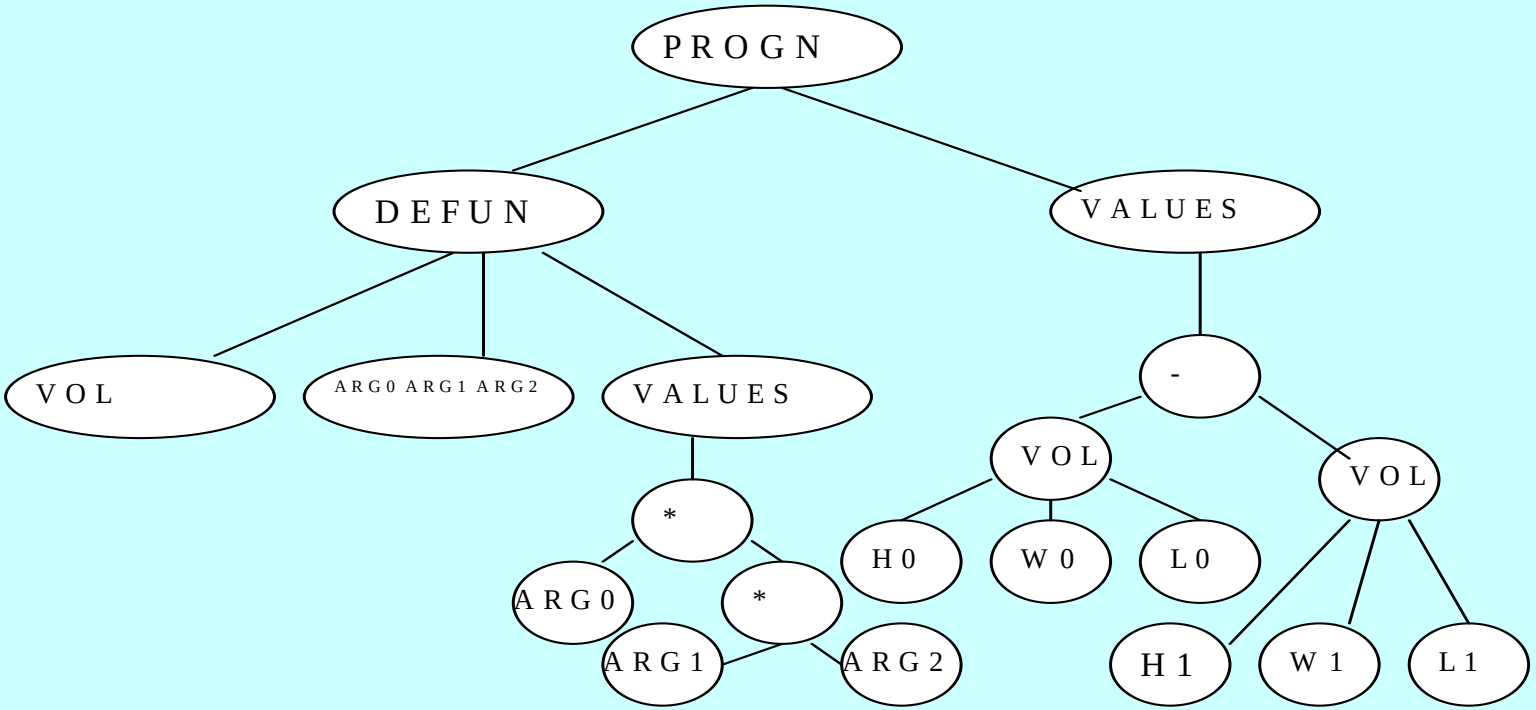
ENFOQUE CON ADF: (PROG (DEF (VOL ARG0 AR1 ARG2)

VALUES (*ARG0 (* ARG1 ARG2))))

(VALUES (- VOL L0 W0 H0)

(VOL L1 W1 H1))))

PROGRAMACIÓN GENÉTICA



PROGRAMACIÓN GENÉTICA

ELEMENTOS DE UN ADF

- NUMEROS DE ADFs
- SUS ARGUMENTOS
- LAS REFERENCIAS JERARQUICAS
- OPERADORES
  - CREACION: ADF O INVOCACION
  - DUPLICACION: ADF O INVOCACION
  - ELIMINACION: ADF O INVOCACION

PROGRAMACIÓN GENÉTICA

ADL (AUTOMATICALLY DEFINED LOOP)

REPETICION DE OPERACIONES PREESTABLECIDAS

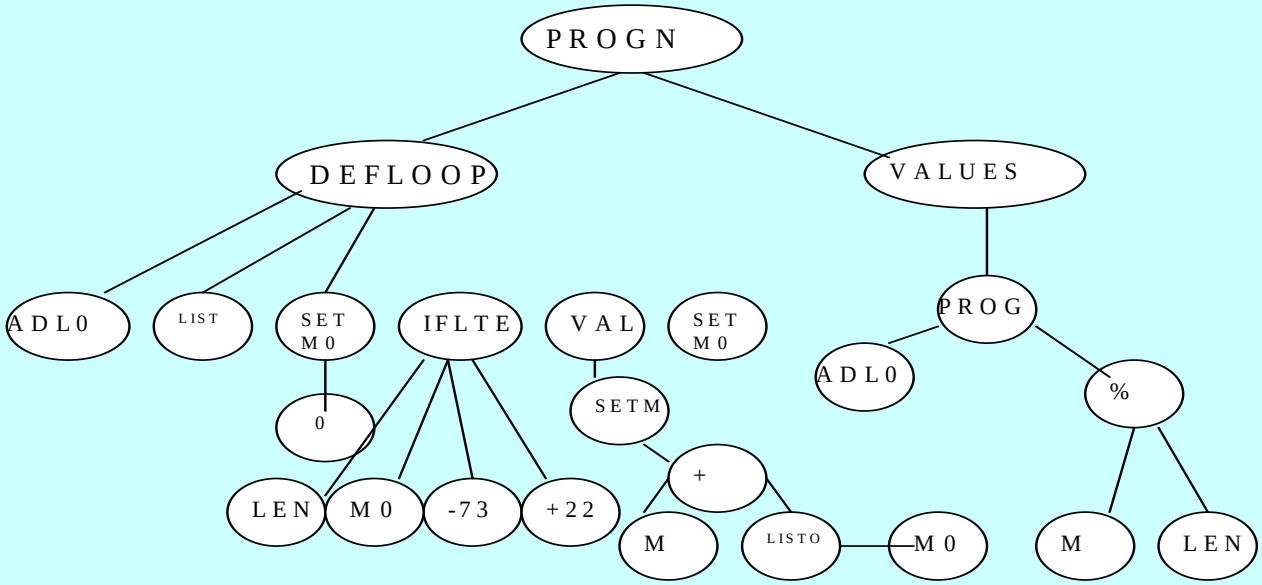
```
for (i=0; i<len; i++)
    m=m+v[i];
```

```
for (LIB; LCB; LUB)
    LBB;
```

PROGRAMACIÓN GENÉTICA

ADL

- rama de inicio lazo
- rama con condición del lazo
- rama con cuerpo
- rama con actualización lazo



PROGRAMACIÓN GENÉTICA

ADL

```
M0=0;
for (i=0; i<LEN;i++
    M0=M0+V[i]
AVERAGE=M0/LEN
```

Pueden haber múltiples ADL en un programa (no anidados entre ellos)

No poseen argumentos y pueden llamar a ADF

PROGRAMACIÓN GENÉTICA

ADL

Operadores

Creación

=> Tomar un pedazo del código y crear ADL

Eliminación

=> regeneración aleatoria del pedazo

=> modificación conjunto terminal y funciones

Duplicación

=> Escoger un ADL existente y modificar nombre, etc.

=> Llamados al ADL viejo son aleatoriamente modificados

PROGRAMACIÓN GENÉTICA

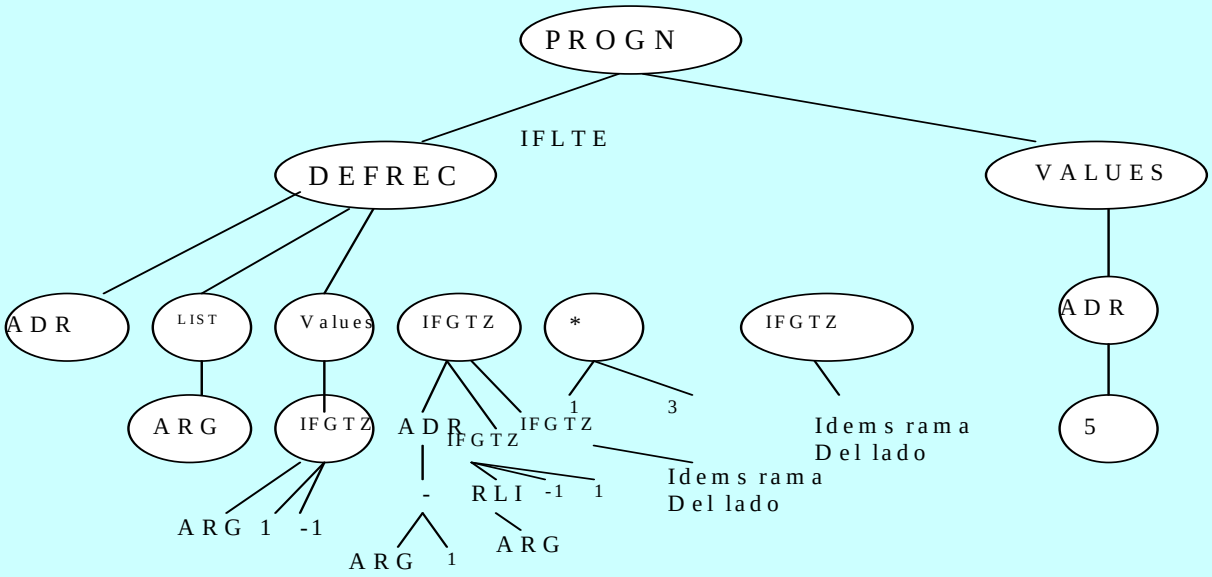
ADR

Partes

- Rama Condicional (RCB)
- Rama Actualizacion (RUB)
- cuerpo (RBB)
- Rama de desarrollo (RGB)

```
float ADR (float ARG)
{   if (RCB (ARG) > 0)
    {       result=RBB(ARG);            /* llamada a ADR*/
            RUB(ARG); }
    else
            result=RGB(ARG);
return(result); }
```

PROGRAMACIÓN GENÉTICA

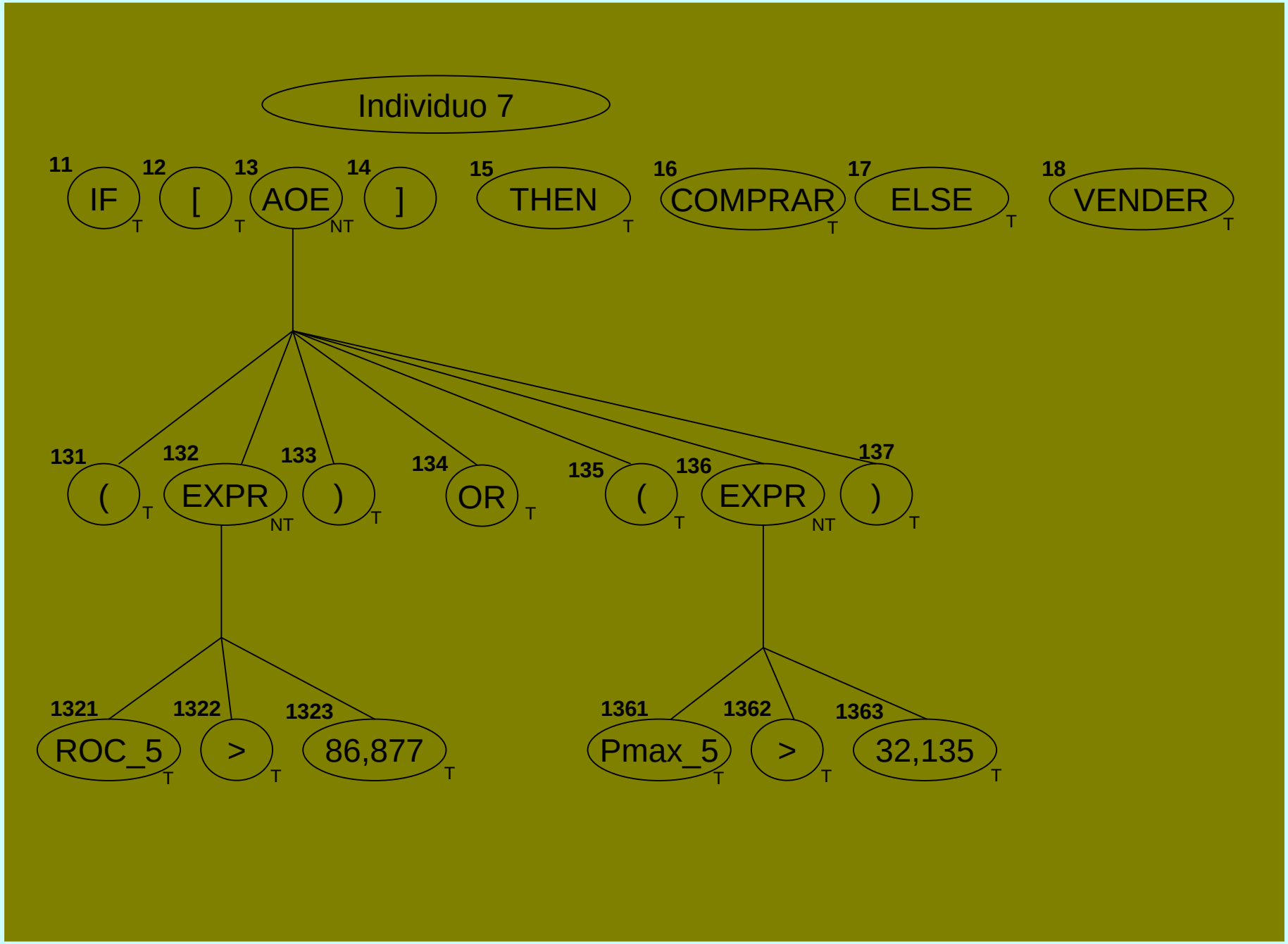


Gramática de una regla (ADR)

IF { ANTECEDENTE } THEN
{ CONSECUENTE }
ELSE
{not- CONSECUENTE }

ADR

IF [(ROC_5 > 86,8) OR (Pmax_5 > 32,1)]
THEN COMPRAR ELSE VENDER


Individuo 7
11
IF
T
12
[
T
13
AOE
NT
14
]
15
THEN
T
16
COMPRAR
T
17
ELSE
T
18
VENDER
T
131
(
T
132
EXPR
NT
133
)
T
134
OR
T
135
(
T
136
EXPR
NT
137
)
T
1321
ROC_5
T
1322
>
T
1323
86,877
T
1361
Pmax_5
T
1362
>
T
1363
32,135
T

PROGRAMACIÓN GENÉTICA

ADR

Operadores

Creación

=> Tomar un pedazo del código y crear ADR

Eliminación

=> regeneración aleatoria del pedazo

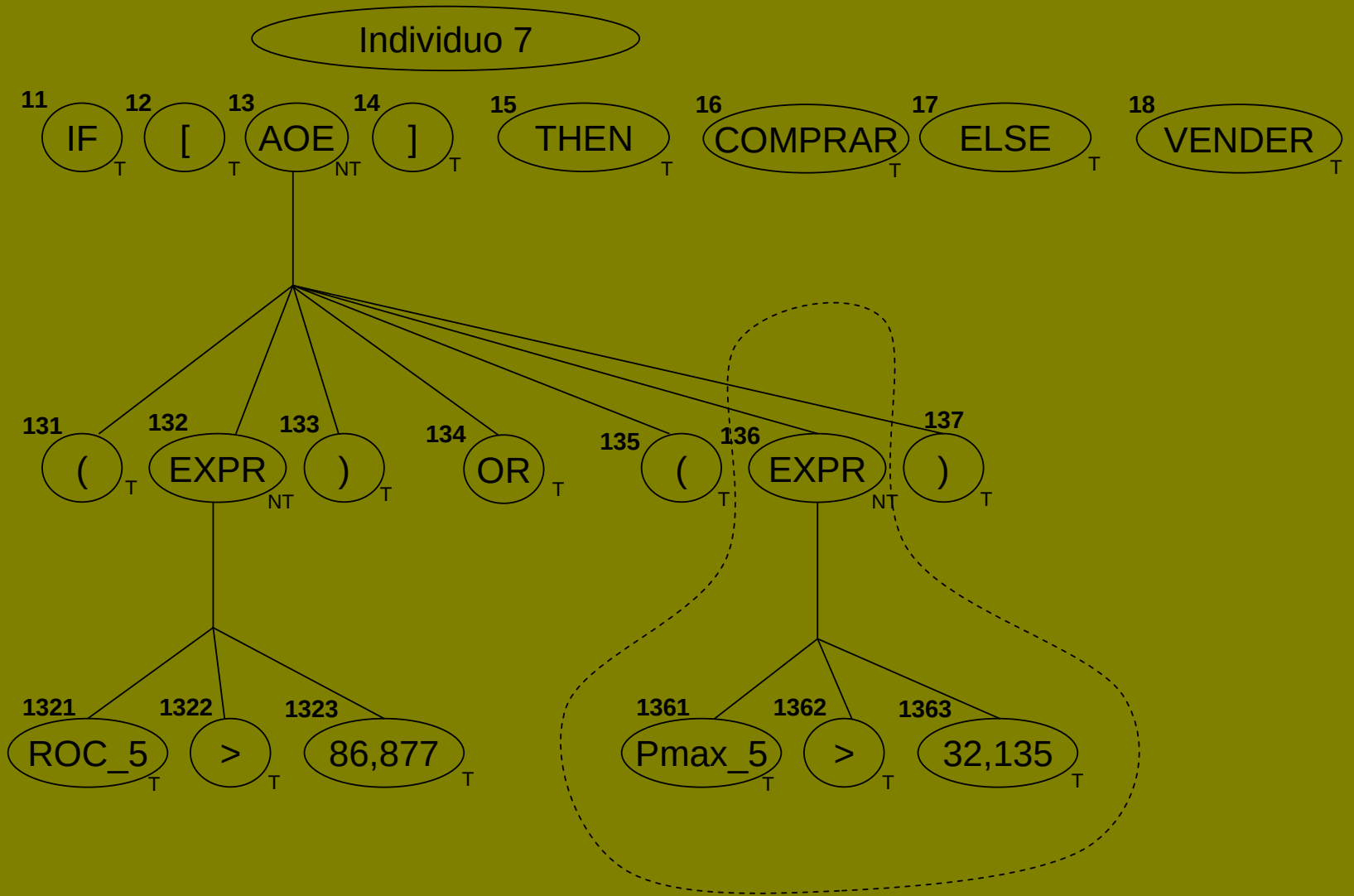
=> modificación conjunto terminal y funciones

Duplicación

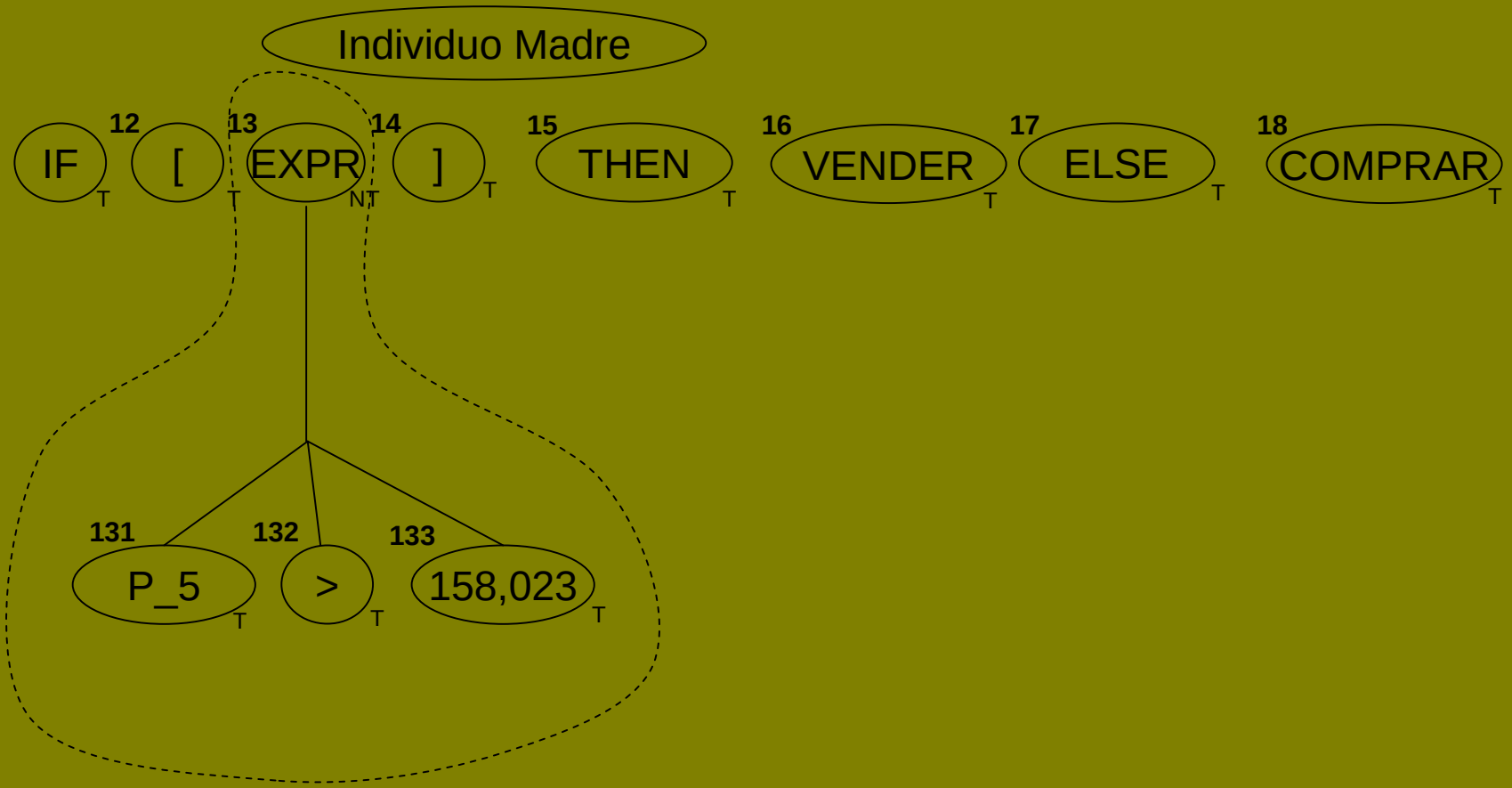
=> Escoger un ADL existente y modificar nombre, etc.

=> Llamados al ADL viejo son aleatoriamente modificados

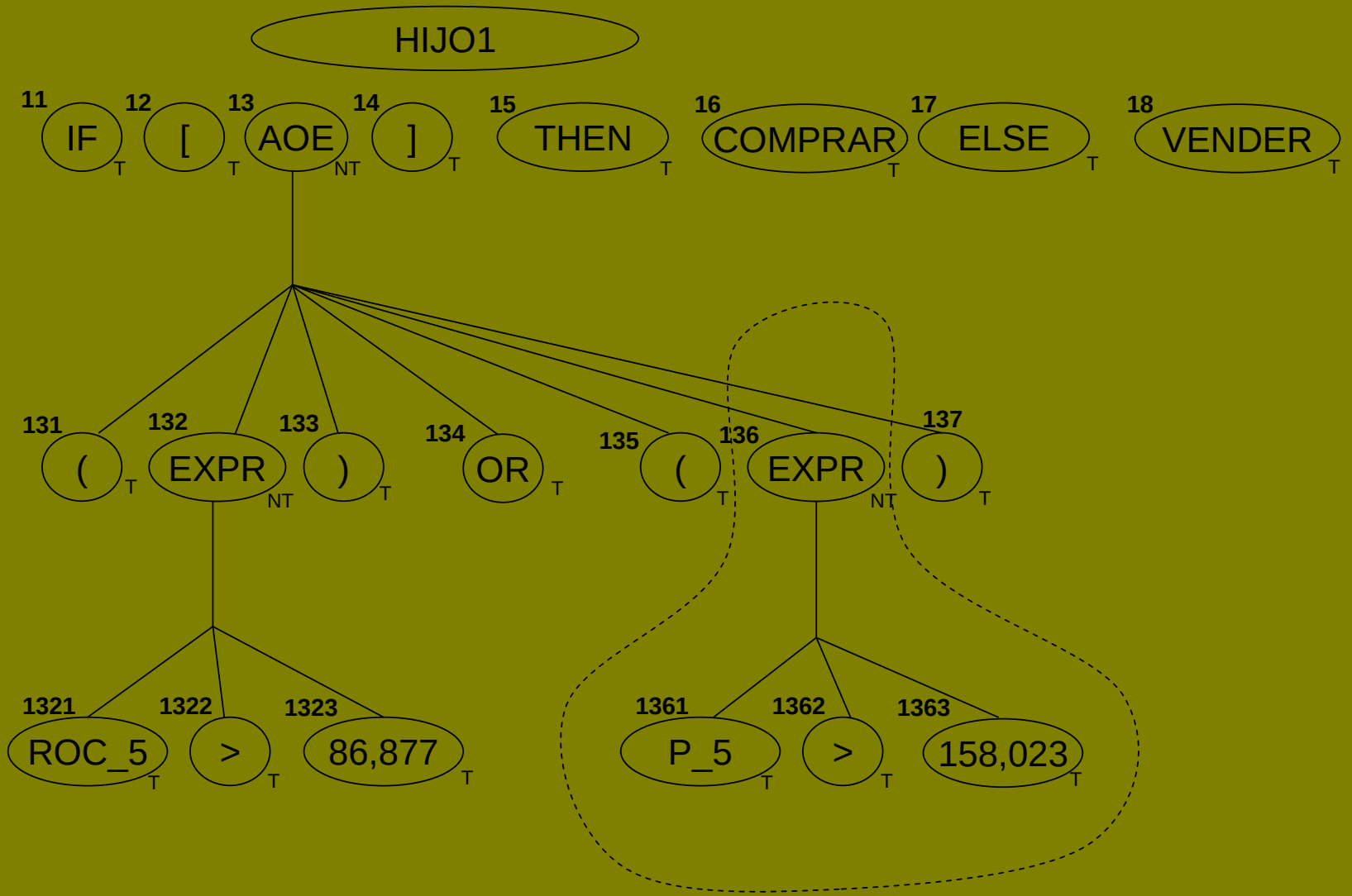
Individuo Padre: IF [(ROC_5 > 86,877) OR (Pmax_5 > 32,135)]
THEN COMPRAR ELSE VENDER



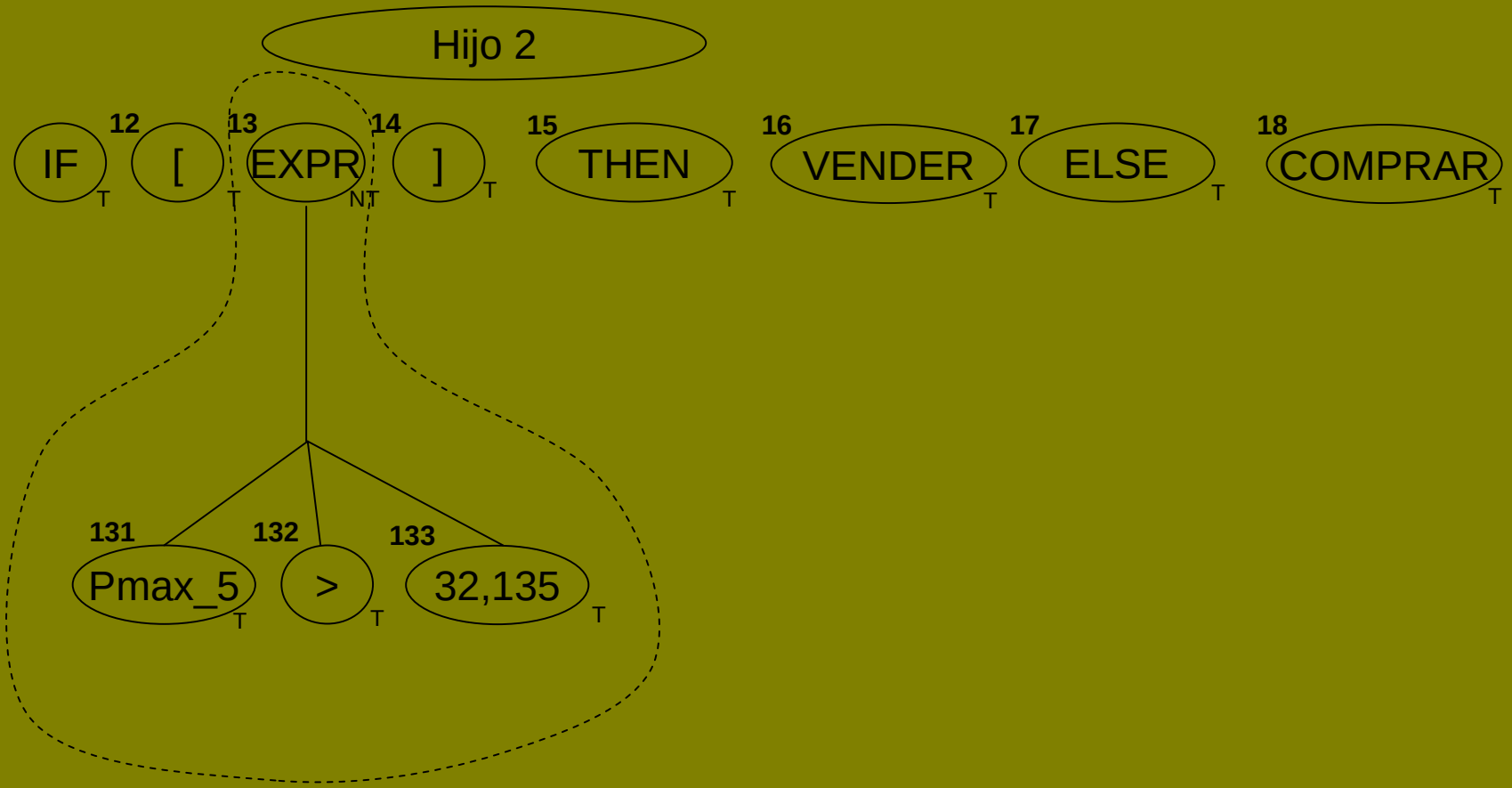
Individuo Madre: IF [P_5 > 158,023] THEN VENDER ELSE COMPRAR



Individuo Hijo1: IF [(ROC_5 > 86,877) OR (P_5 > 158,023)] THEN COMPRAR ELSE VENDER



Individuo Hijo2: IF [Pmax_5 > 32,135] THEN VENDER ELSE COMPRAR



PROGRAMACIÓN GENÉTICA

AREAS DE APLICACIÓN

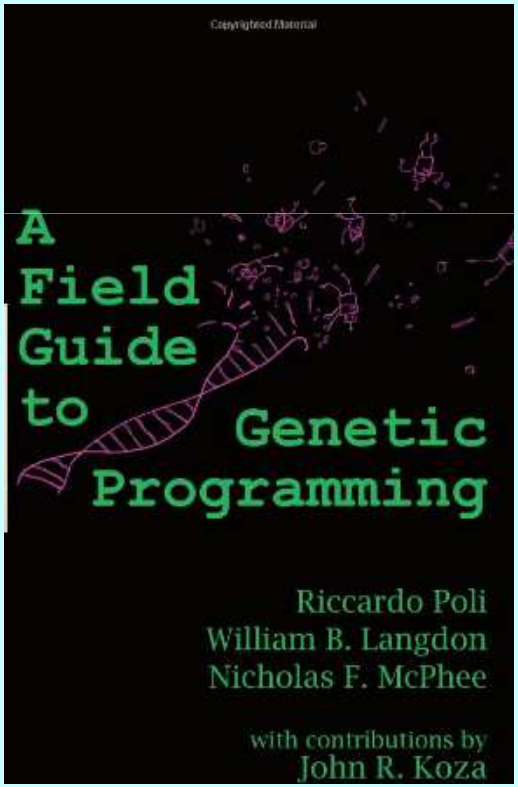
- CONTROL: IDENTIFICACION, ETC.
- MINERIA DE DATOS
- SISTEMAS MULTIAGENTES
- HARDWARE EVOLUTIVO

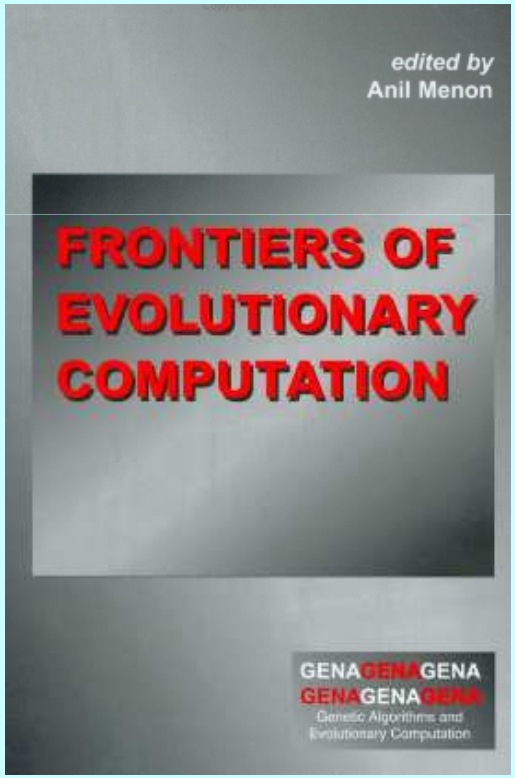
www.aic.nrl.navy.mil/galist

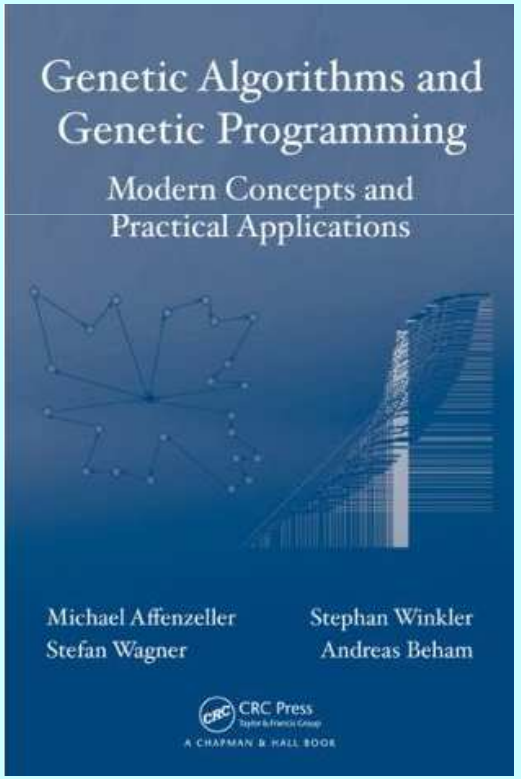
Kernel GPC++

Es una biblioteca de clases escritas en el lenguaje de programación orientado a objetos C++, que permite utilizar la técnica de Programación Genética con un esfuerzo mínimo, dado que el *kernel* proporciona el conjunto de estructuras, clases y algoritmos necesarios para utilizar la técnica.

Algunas Referencias







PROGRAMACIÓN EVOLUTIVA

PROGRAMACIÓN EVOLUTIVA

Evoluciona algoritmo que opera secuencia de simbolos tomados de un alfabeto finito

Simbolo de salida maximiza rendimiento si algoritmo predice proximo simbolo

Enfasis en el cambio de comportamiento

Por ejemplo: un conjunto de *maquinas de estado finito* se expone a un ambiente.

PROGRAMACIÓN EVOLUTIVA

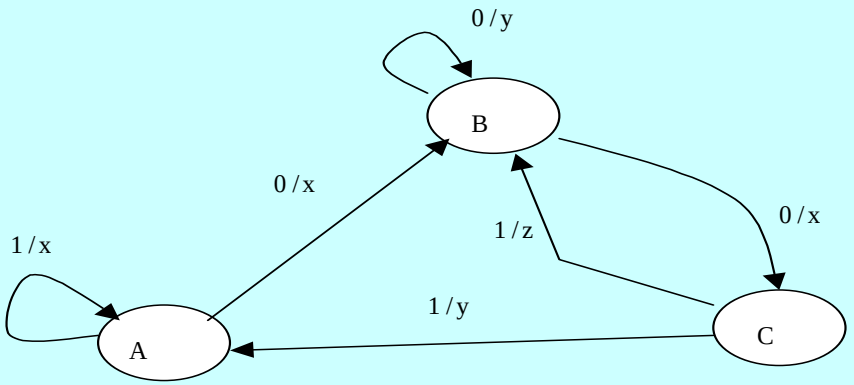
Para cada maquina de Estados Finitos, su predicción es medida con respecto al ambiente, como una función error.

Después que se hace la ultima predicción, un error promedio para cada maquina es calculado para indicar la aptitud de ella.

Las maquinas que presenten el mejor resultado (mínimo error) se retienen para ser padres en la próxima generación.

PROGRAMACIÓN EVOLUTIVA

Edo. Presente	C	B	C	A	A	B
Entrada	0	1	1	1	0	1
Prox.	B	C	A	A	B	C
Salida	x	z	y	x	x	z



PROGRAMACIÓN EVOLUTIVA

MACROALGORITMO

1. DEFINIR SECUENCIA DE SIMBOLOS Y FA
2. INICIAR POBLACION MEF
3. INTRODUCIR SIMBOLO ENTRADA Y VER SU SALIDA
4. EVALUAR SI PREDIJO BIEN
5. CREAR NUEVOS HIJOS USANDO MUTACION: CAMBIAR SIMBOLOS, ESTADO DE TRANSICION, O EDO. INICIAL, ANADIR O ELIMINAR SIMBOLO
6. SELECCIONAR MEJORES PADRES
7. REGRESAR A 3

Esquema de generación de descendientes

Por cada maquina padre se obtiene un descendiente aplicando sobre esta una mutación aleatoria:

- Cambiar un símbolo de salida.
- Cambiar un estado de transición.
- Agregar un estado.
- Eliminar un estado.
- Cambiar el estado inicial.

PROGRAMACIÓN EVOLUTIVA

EJEMPLO: PREDICCION DE TIEMPO

{SOL, LLUVIA, SOL, LLUVIA, TORMENTA, ...}

REPRESENTADO POR {0, 1, 0, 1, 2, ...}

MATRIZ DE PAGO

	0	1	2
0	1	-10	-100
1	-10	5	-40
2	-25	-5	10

Estrategias Evolutivas

ESTRATEGIAS EVOLUTIVAS

Son métodos estocásticos con paso adaptativo, que permiten resolver problemas de optimización paramétrica.

A estos métodos se le han incorporado procedimientos propios de la CE que los han convertido en un paradigma mas de dicha metodología.

ESTRATEGIAS EVOLUTIVAS

Las EEs pueden definirse como algoritmos evolutivos enfocados hacia la optimización paramétrica que utilizan una representación a través de vectores reales, una selección determinista, operadores específicos de mutación y cruce (Cruce Numerico) y restricciones implicitas en la representacion u operadores.

Las estrategias evolutivas pueden dividirse en:

- Simples.
- Múltiples.

ESTRATEGIAS EVOLUTIVAS

Las EE simples son consideradas como procedimientos estocásticos de optimización paramétrica con paso adaptativo (similares al temple simulado).

Se hace evolucionar un solo individuo usando únicamente un operador genético, la mutación.

ESTRATEGIAS EVOLUTIVAS

Se denotan como (1+1)-EE:

- El individuo (v) se representa a través de un par de vectores reales v = (x;s).
- El vector *x* representa una solución, y *s* es un vector de desviaciones típicas usado para la mutación.

$$x[t+1] = \begin{cases} x[t] + N(0, s[k]) & \text{ssi } x[t+1] > x[t] \text{ y } x[t+1] \in \psi \\ x[t] & \text{de lo contrario} \end{cases}$$

  - N(0,s): Representa un vector de números aleatorios gausianos independientes con medias 0 y un vector de desviaciones típicas *s*, donde cada uno de los elementos del vector *s* es la desviación típica a usar por cada uno de los elementos del vector *x*.
  - ψ: Representa el espacio de soluciones.

ESTRATEGIAS EVOLUTIVAS

Las EEs múltiples surgen como respuesta a las debilidades de las EEs simples.

En las EEs múltiples existen múltiples individuos (población), y se producen en cada generación varios nuevos individuos, usando tanto mutación como cruce.

ESTRATEGIAS EVOLUTIVAS

Se denotan como (λ^{+}, μ)-EE, donde λ es el tamaño de la población y μ es el tamaño de la descendencia.

Los símbolos (+ ,) representan dos posibles mecanismos de reemplazo:

Por inclusión (+): padres e hijos compiten

Por inserción (,): solo sobreviven hijos y compiten entre ellos

ESTRATEGIAS EVOLUTIVAS

Operadores de cruce y mutacion:

Cruce intermedio:

Padres: $\left\{ \begin{array}{l} \langle X1, ..., Xm \rangle, \langle S1, ..., Sm \rangle \\ \langle Y1, ..., Ym \rangle, \langle S'1, ..., S'm \rangle \end{array} \right.$

resultado: $\langle 1/2\,(X1+Y1), ..., 1/2(Xm+Ym);$

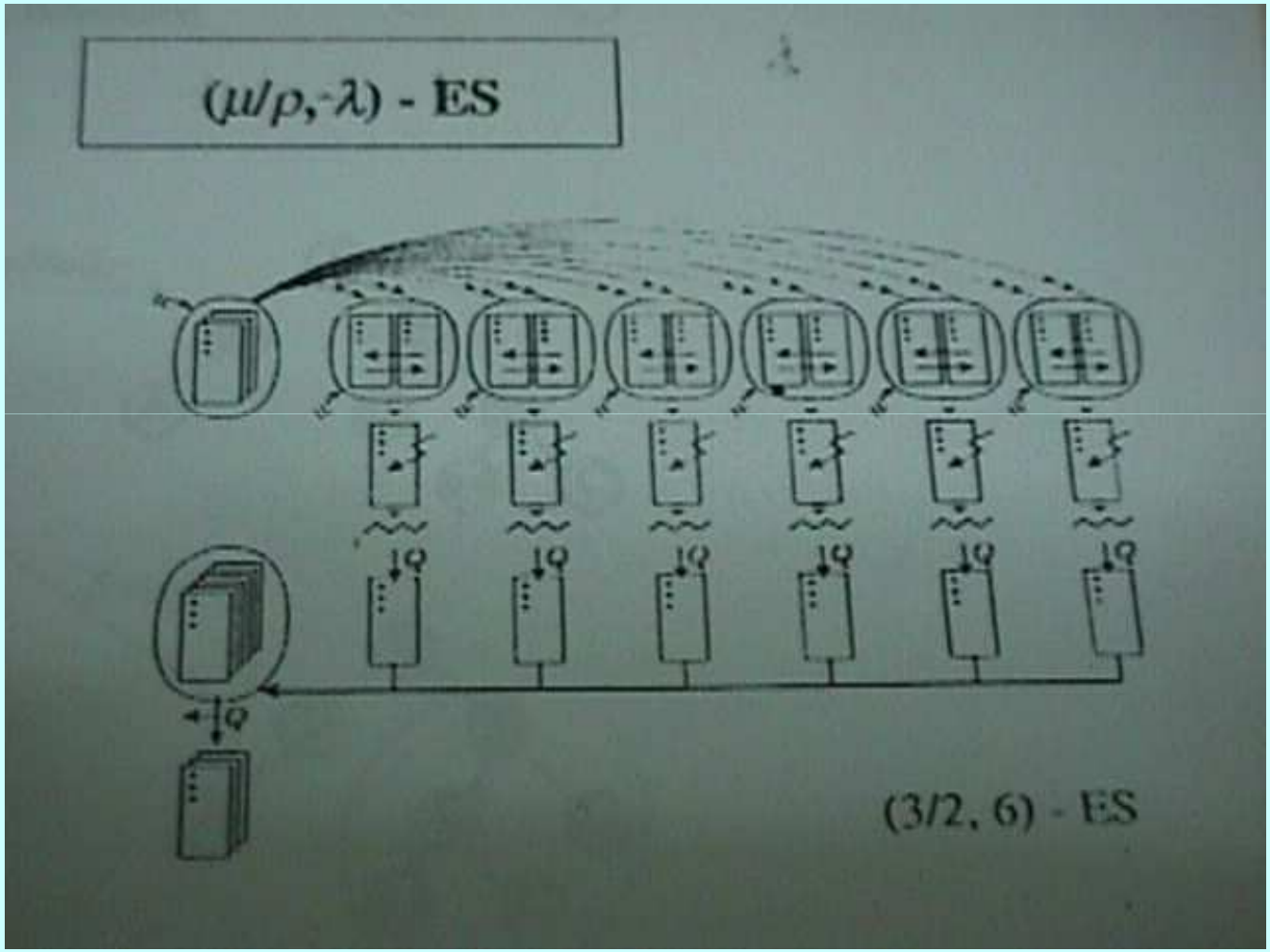
$\mathrm{sqrt}(S1^2+S'1^2), ..., \mathrm{sqrt}(S1^2+S'1^2)\rangle$

ESTRATEGIAS EVOLUTIVAS
Anidamiento

$[u'+, \lambda'(u+, \lambda)^{y}]^{y'}ES$

Hay *u'* poblaciones de *u* padres. Ellas son usadas para generar *λ'* poblaciones iniciales de *u* individuos. Por cada *λ'* población un $(u+, \lambda)^{y}ES$ es ejecutado durante *y* generaciones. Este esquema es respetido *y'* veces

ESTRATEGIAS EVOLUTIVAS



ESTRATEGIAS EVOLUTIVAS

- **RESOLUCIÓN DEL PROBLEMA DL CUBO.**

  - OBJETIVO: TODOS LOS LADOS CON EL MISMO COLOR

  - FUNCION: Fext= Fbas + Fborde + Fesquina

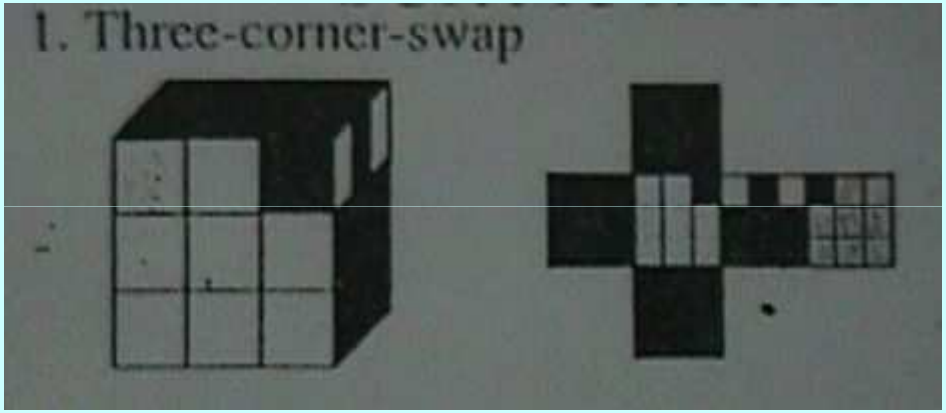
  Fbas: COLOR ERRADO DE LADO => +1

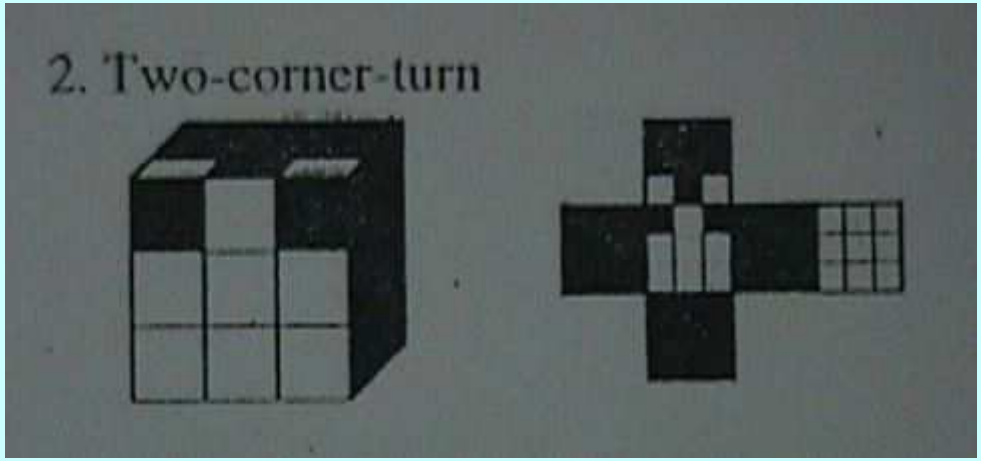
  Fesquina: COLOR ESQUINA ERRADO =>+6

  Fborde: COLOR BORDE ERRADO=>+4

ESTRATEGIAS EVOLUTIVAS

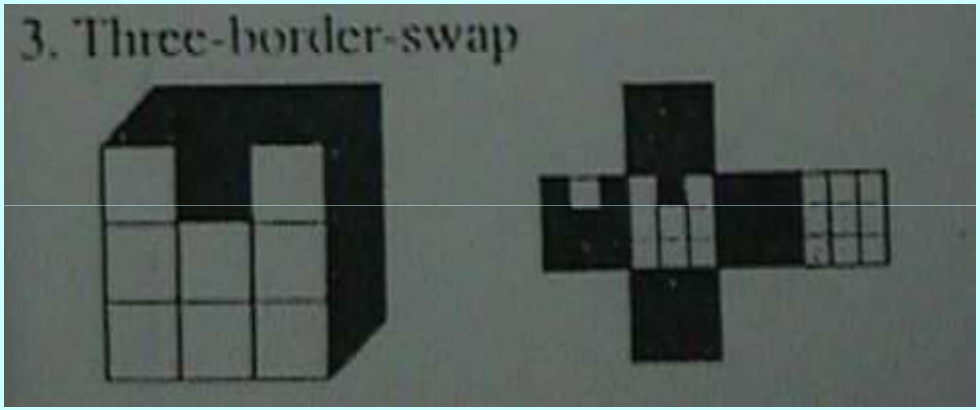
- **OPERADOR MUTACIÓN:**

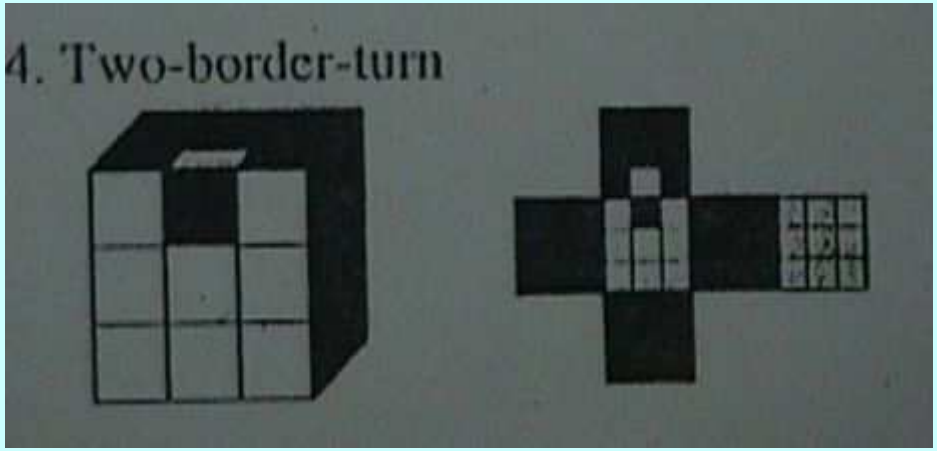




ESTRATEGIAS EVOLUTIVAS

- **OPERADOR MUTACIÓN:**





ESTRATEGIAS EVOLUTIVAS

- **EVOLUCIÓN:**