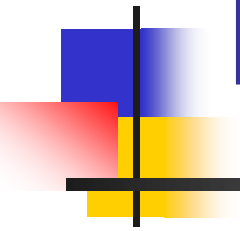
Programación Automática

MÁSTER EN CIENCIA Y TECNOLOGÍA INFORMÁTICA

Ricardo Aler Mur

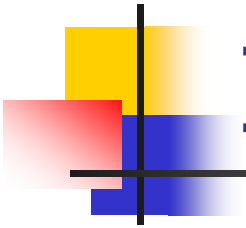


Programación Automática

Ricardo Aler Mur

Universidad Carlos III de Madrid

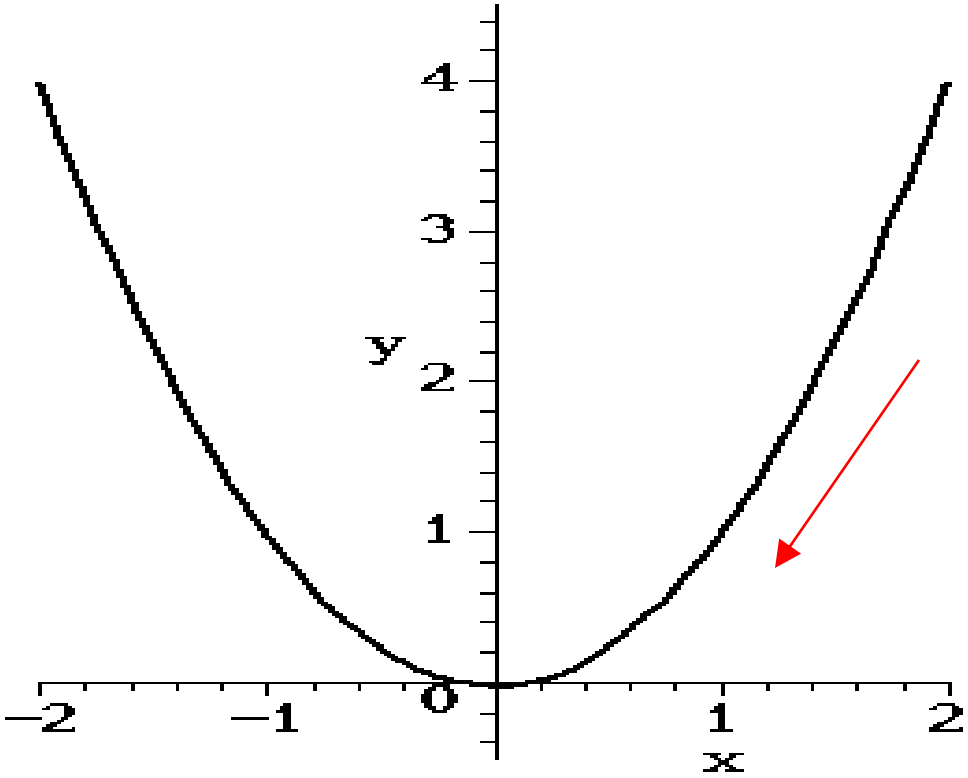
http://www.uc3m.es/uc3m/dpto/INF/aler



Índice

1. Optimización por gradiente vs. Búsqueda genética
2. Estimation of Distribution Algorithms
3. Probabilistic Incremental Program Evolution
4. Estimation of Distribution Programming
5. Otros (extended competent genetic programming, estimación de gramáticas, etc.)
6. ADATE (programación funcional inductiva)

Idea de optimización por gradiente (valores continuos)



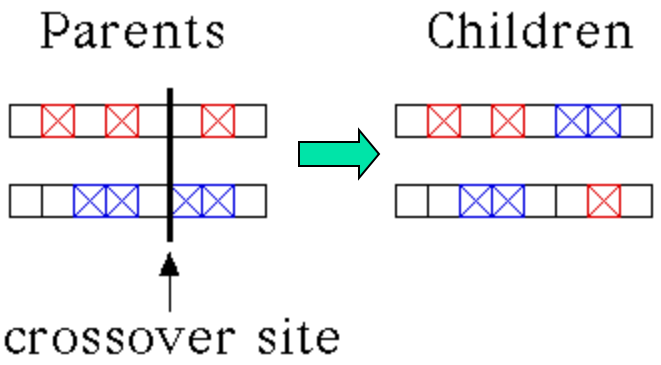
Información para la búsqueda: el gradiente

Idea de búsqueda genética

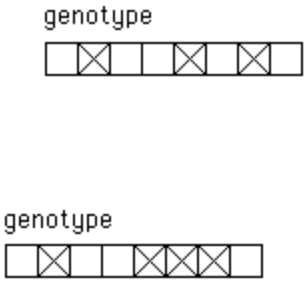
- Posible significado de la población: si hay 90 de 100 individuos con el primer bit a 1, es que ese bit es bueno "en un 0.9" independiente de lo que valga el resto
- Building blocks: 1***, 01**, 1**0
- Es decir, la población mantiene una especie de distribución de probabilidad de que el primer bit valga 1, de que el primero y el segundo valgan 01, etc.

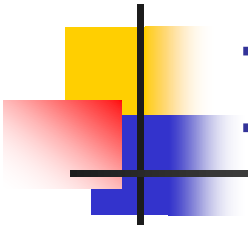
Operadores genéticos

Cruce



Mutación





Idea de búsqueda genética

- La población (distribución de probabilidad) se va actualizando con los operadores genéticos
- Selección: 010010 va a tener muchos hijos
  - Hacer más probables las combinaciones de bits de los buenos individuos
- Mutación: 010010 -> 010011
  - Explorar regiones adyacentes al individuo bueno
- Cruce: es como una macro-mutación, controlada por el resto de la población

Estimation of Distribution Algorithms

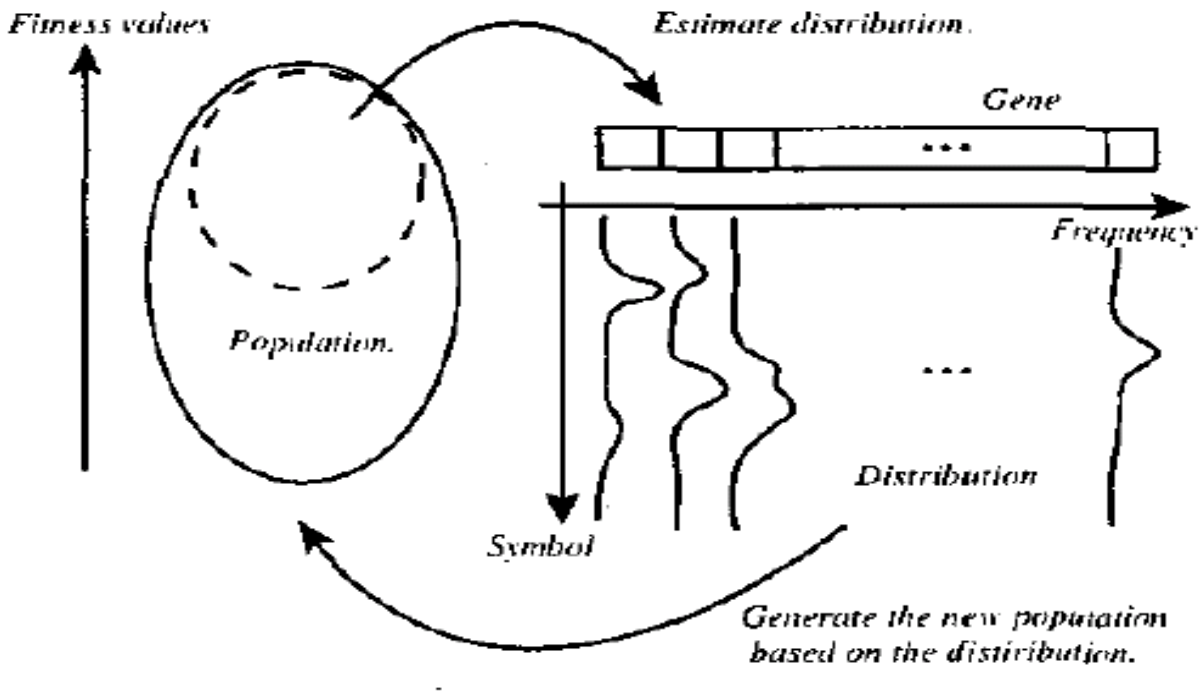
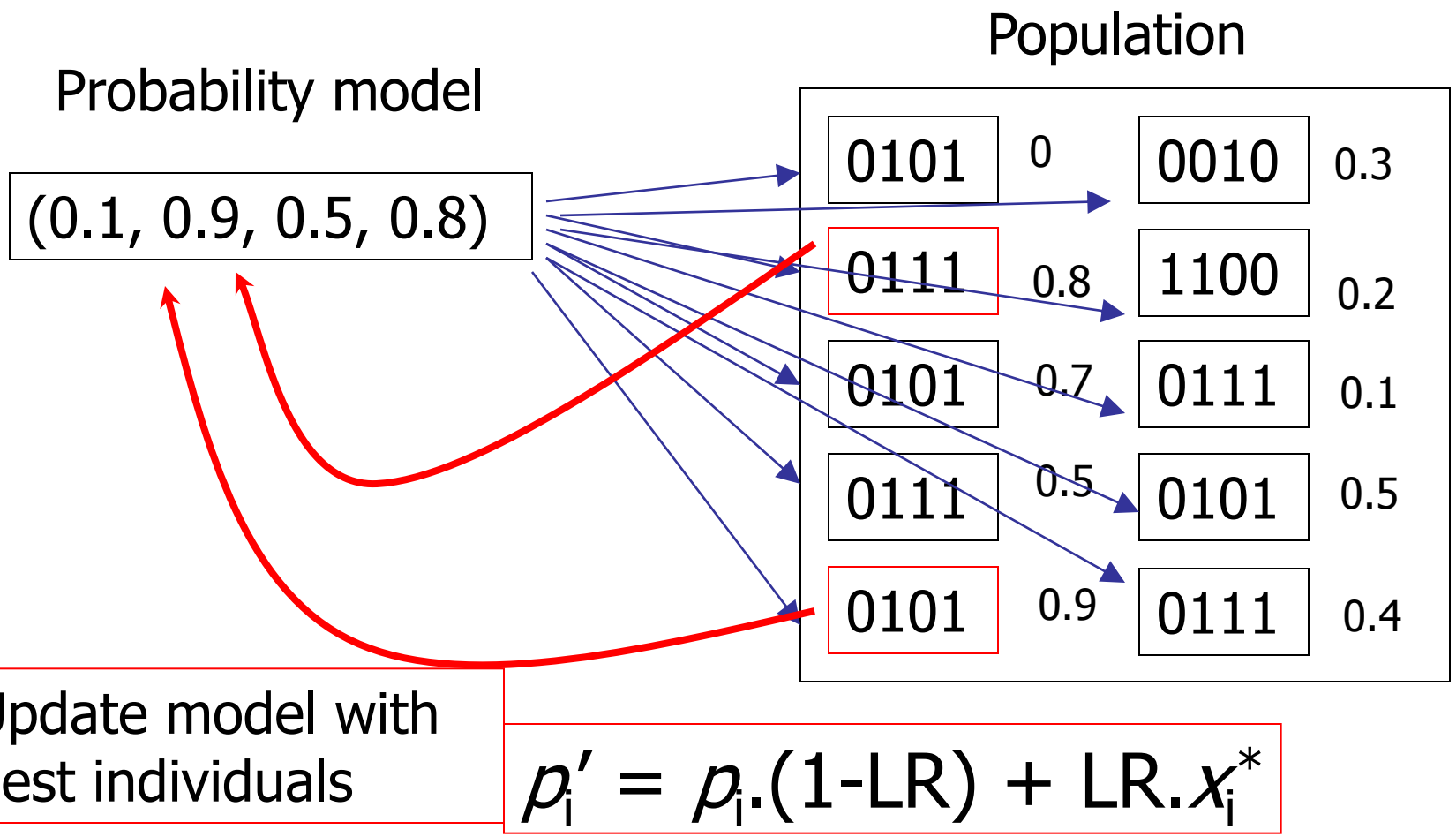


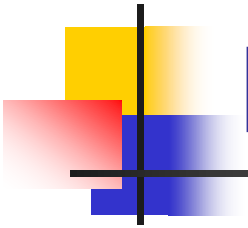
Figure 1: EDA's algorithm.

Fuente: Yanay e Iba, 2003

Population Based Incremental Learning PBIL [Baluja, 94]

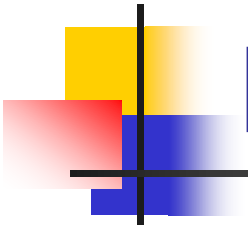


Population Based Incremental Learning [Baluja, 94]



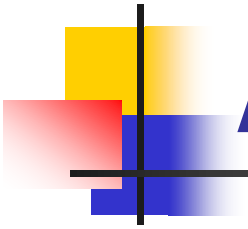
- Aprende **explícitamente** un modelo probabilístico de las zonas "interesantes" del espacio de búsqueda
- Modelo probabilístico:
  - Vector $p=(p_1, p_2, ..., p_n)$
  - p_i indica la probabilidad de generar un 1 en la posición *i* del cromosoma $x=(x_1, x_2, ..., x_n)$
  - Inicialmente $p=(0.5, 0.5, ..., 0.5)$

Population Based Incremental Learning [Baluja, 94]



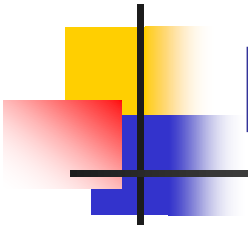
- Se genera una población de x, se calcula su fitness y se actualiza p con los mejores M individuos
- Actualización: $p_i' = p_i.(1\text{-}LR) + LR.x_i^*$
- Al final, p debería converger a una solución:
  - p=(0.99, 0.001, ..., 0.99)

Estimation of Distribution Algorithms (EDA's)



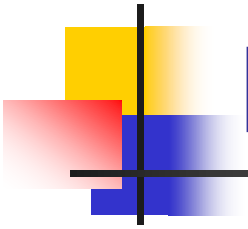
1. Generar población inicial
2. Seleccionar un número de los mejores individuos
3. Estimar la distribución
4. Generar una nueva población a partir de la distribución (en ocasiones, se mezcla la población original con la nueva)
5. Si no se termina, volver a 2

Estado de la cuestión GP-EDA's



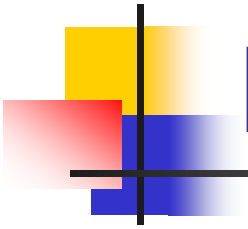
- GP-EDA's: aplicación de algoritmos de estimación de distribuciones a la Programación Genética
- "A Survey of Probabilistic Model Building Genetic Programming". Yin Shan, Robert McKay, Daryl Essam, Hussein Abbass. TR-ALAR-200510014

Probabilistic Incremental Program Evolution (PIPE)



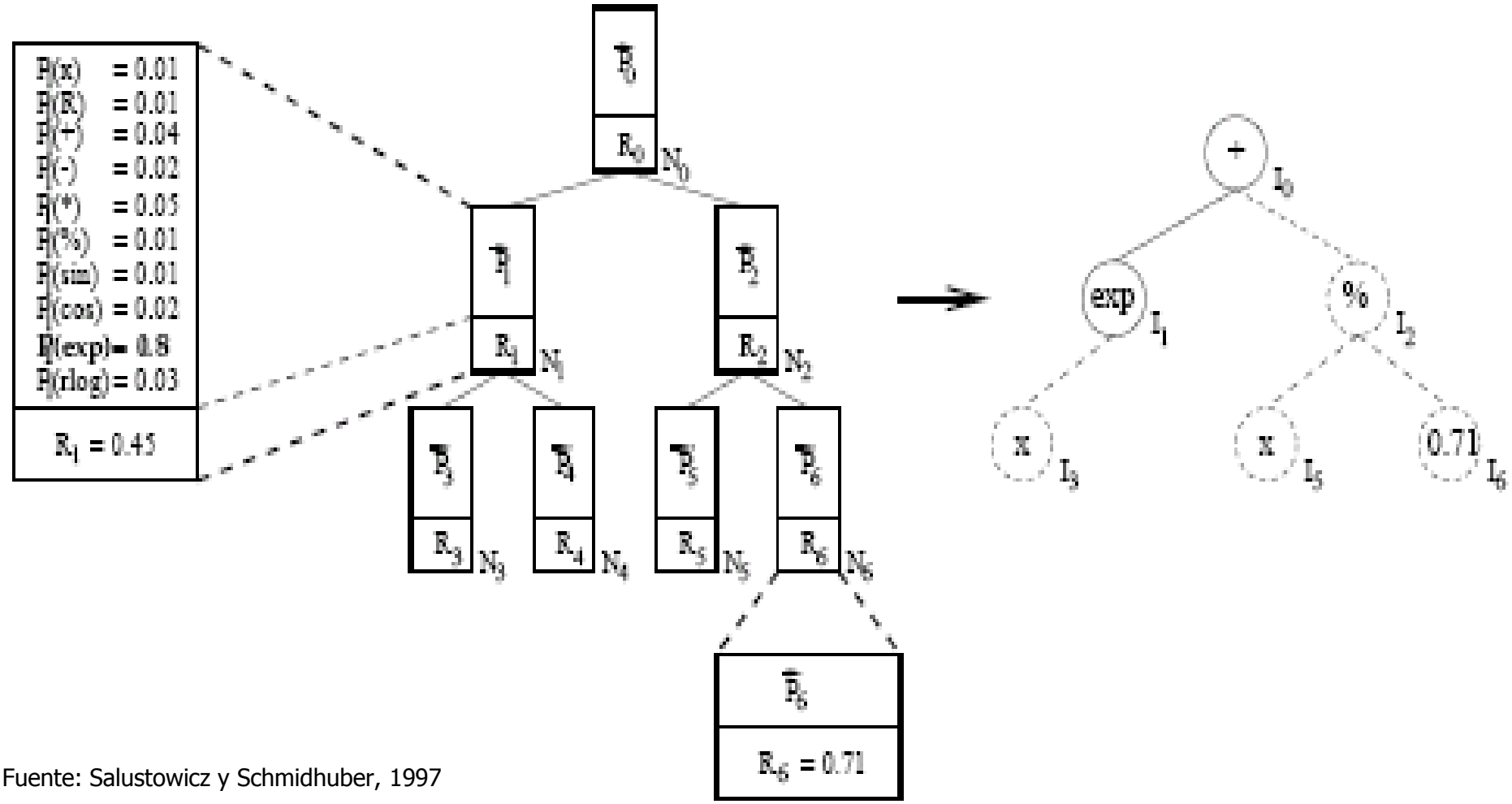
- Aplicación de algoritmos EDAs para la evolución de programas
- PIPE [Salustowicz, Schmidhuber, 97]
- Evolución de árboles (parse trees)
- Búsqueda en el espacio de distribuciones de probabilidad de árboles
- Busca encontrar una distribución que genere buenos árboles

Probabilistic Incremental Program Evolution (PIPE)



- Los programas tienen:
  - Funciones: $F = \{F_1, ..., F_k\}$
  - Terminales: $T = \{T_1, ..., T_l\}$
- La Generic Random Constant (GRC):
  - Similar a la Ephemeral Random Constant (ERC)
  - Cuando se la utiliza en la creación de programas, o bien toma un valor real aleatorio, o bien toma el valor fijo almacenado
- Al igual que en PG, se respeta la *closure*

Probabilistic Prototype Tree (PPT)



Fuente: Salustowicz y Schmidhuber, 1997

Nodo inicial del PPT

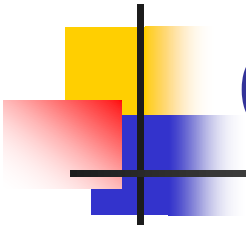
Inicialmente: $P_j(I) := \frac{P_T}{l}, \forall I : I \in T$

$$P_j(I) := \frac{1 - P_T}{k}, \forall I : I \in F,$$

$P_j(x)$	$= 0.3$
$P_j(R)$	$= 0.3$
$P_j(+)$	$= 0.05$
$P_j(-)$	$= 0.05$
$P_j(*)$	$= 0.05$
$P_j(\%)$	$= 0.05$
$P_j(\sin)$	$= 0.05$
$P_j(\cos)$	$= 0.05$
$P_j(\exp)$	$= 0.05$
$P_j(\mathrm{rlog})$	$= 0.05$
R_j	$= 0.45$

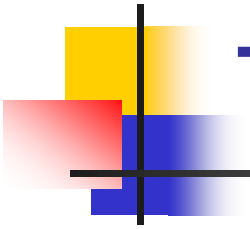
N_j

Fuente: Salustowicz y Schmidhuber, 1997



Creación de la población inicial

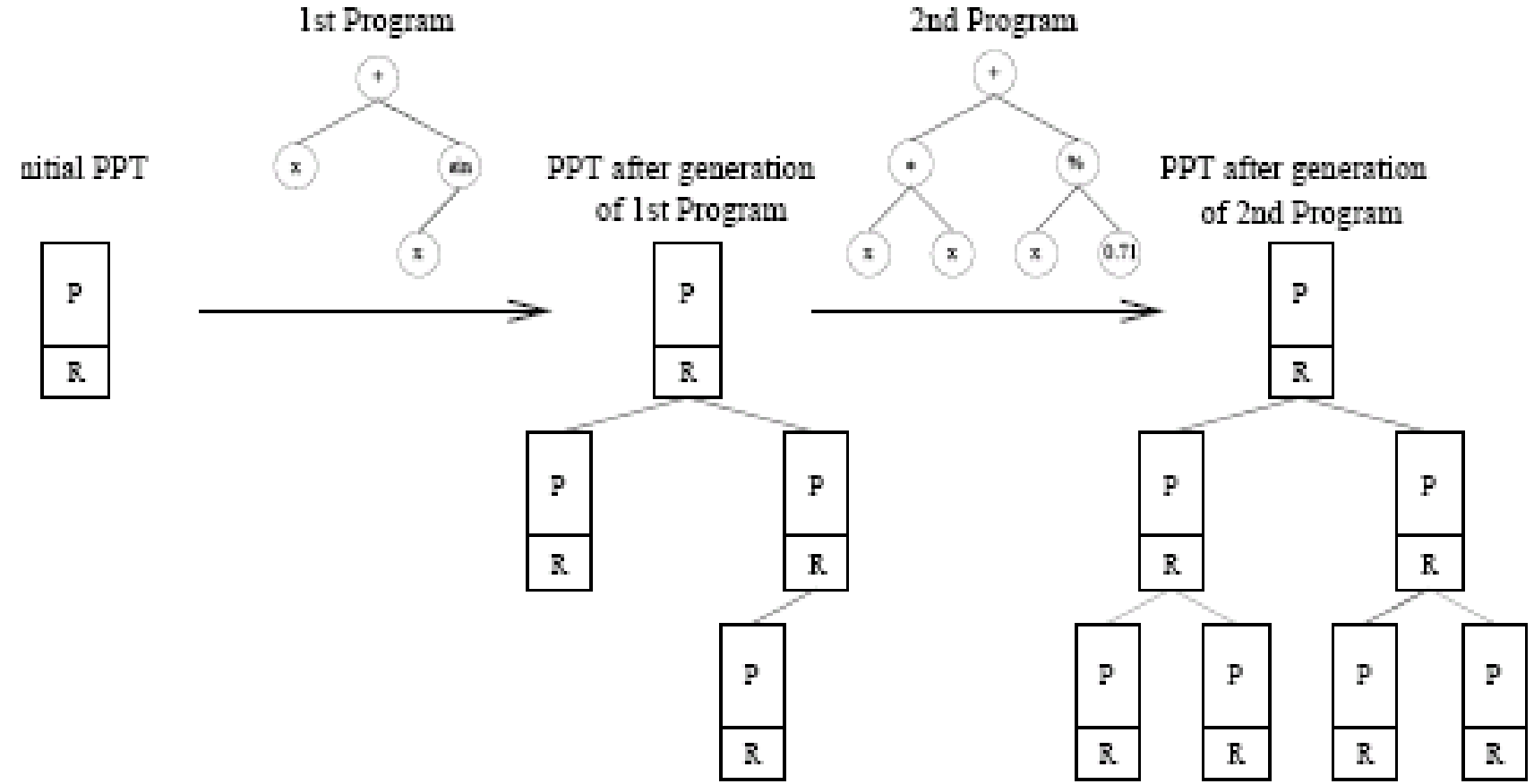
- Se recorre el PPT de arriba abajo (comenzando en la raíz) y de izquierda a derecha)
- Se selecciona un símbolo del nodo según su probabilidad
- Si se selecciona R (la GRC), entonces:
  - Si la prob(R) > threshold, entonces R
  - Si no, generar aleatoriamente un valor para R



Tamaño del PPT

- Experimentalmente, es suficiente con mantener un PPT 3 veces*nodos a la mejor solución encontrada
- Inicialmente, el PPT contiene sólo el nodo raíz
- Los nodos se van creando cuando son necesarios (cuando en un nodo hoja del PPT se selecciona una función, es necesario crear sus argumentos)

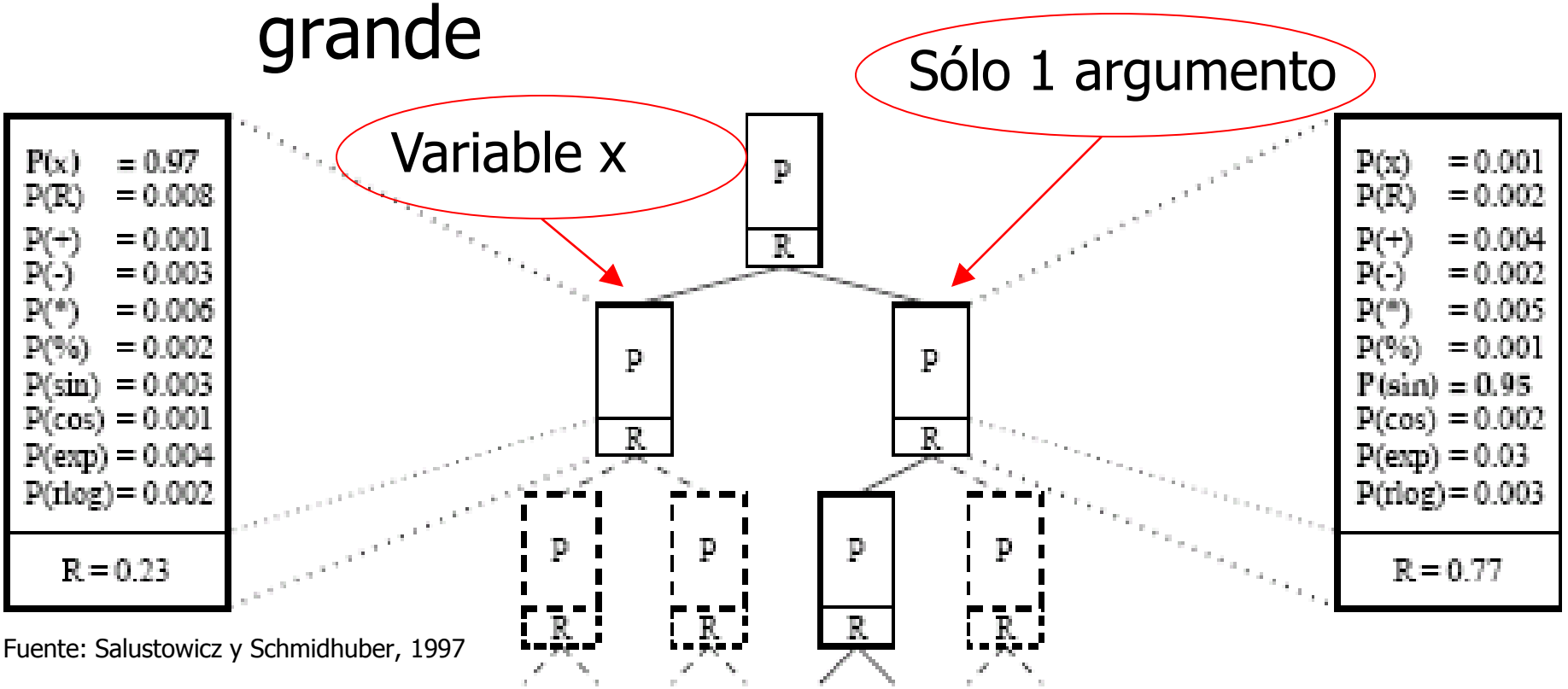
Crecimiento del PPT



Fuente: Salustowicz y Schmidhuber, 1997

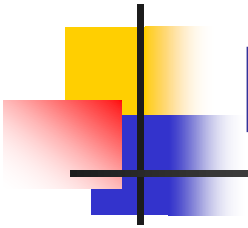
Poda del PPT

- En caso de que en un nodo, uno de los símbolos tenga una probabilidad muy grande

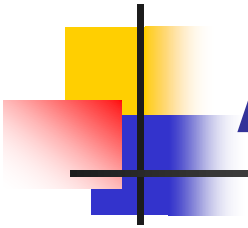


Fuente: Salustowicz y Schmidhuber, 1997

Probabilistic Incremental Program Evolution (PIPE)

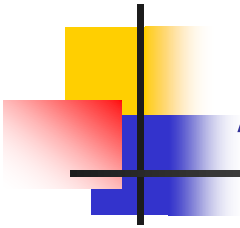


- Se almacena una distribución de probabilidad sobre el conjunto de programas (PPT)
- Cada generación, se incrementa la probabilidad de que el mejor programa sea generado (actualizando el PPT)
- Se minimiza.
- A igualdad de *fitness*, se prefiere el programa pequeño
- Evaluación paralela de programas, terminar cuando aparezca el mejor



Algoritmo PIPE

- Utiliza dos formas de aprendizaje:
  - *Generation based learning (GBL):* actualiza el PPT en dirección al mejor programa encontrado en esa generación
  - *Elitist learning (EL):* actualiza el PPT en dirección al mejor programa encontrado hasta el momento



Algoritmo PIPE

1. Población inicial
2. Evaluación de la población
3. Aprendizaje (adaptación del PPT): con probabilidad P_{el}
    1. Incrementar la prob. de que PPT genere el mejor programa de la generación actual
    2. Incrementar la prob. De que PPT genere el mejor programa encontrado hasta el momento
4. Mutación del PPT (exploración alrededor del mejor)
5. Poda del PPT (ramas con prob. muy baja)

Adaptación del PPT

1. Sea $PROG_b$ el programa hacia el que se quiere adaptar el PPT (el mejor)
2. Se calcula probabilidad de generar $PROG_b$ con el PPT actual

$$P(\text{PROG}_b) = \prod_{j:N_j \text{ used to generate PROG}_b} P_j(I_j(\text{PROG}_b))$$

Adaptación del PPT

3. Se calcula la probabilidad objetivo (la prob. con la que el nuevo PPT debería generar $PROG_b$

$$P_{TARGET} = P(\text{PROG}_b) + (1 - P(\text{PROG}_b)) \cdot lr \cdot \frac{\varepsilon + FIT(\text{PROG}^{el})}{\varepsilon + FIT(\text{PROG}_b)}$$

- *lr* es la "learning rate" (cuanto más alta, más cambia el PPT)
- El cociente controla la magnitud del cambio (recordar que se minimiza)
- Si épsilon, grande, el aprendizaje es independiente de la fitness (el cociente vale 1)

Adaptación del PPT

4. Se iteran los cambios en el PPT, hasta que $P(PROG_b)=P_{TARGET}$ (las probabilidades se actualizan en paralelo). c^{lr} (0.1) controla la precisión final (valor pequeño) y la rapidez (grande)

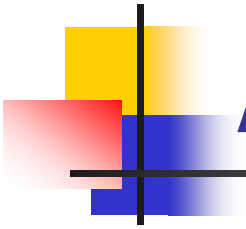
$$\text{REPEAT UNTIL } P(\text{PROG}_b) \geq P_{TARGET} :$$

$$P_j(I_j(\text{PROG}_b)) := P_j(I_j(\text{PROG}_b)) + c^{lr} \cdot lr \cdot (1 - P_j(I_j(\text{PROG}_b)))$$

Adaptación del PPT

5. Renormalización del PPT (se disminuyen las probabilidades de las instrucciones que no están en PROG_b, proporcionalmente a su valor actual)

$$P_j(I) := P_j(I) \cdot \left(1 - \frac{1 - \sum_{I^* \in S} P_j(I^*)}{P_j(I_j(\mathrm{PROG}_b)) - \sum_{I^* \in S} P_j(I^*)} \right) \quad \forall P_j(I) : I \neq I_j(\mathrm{PROG}_b)$$



Adaptación del PPT

6. Se copian las constantes R de $PROG_b$ al PPT

Mutación del PPT

- Para explorar los programas alrededor de $PROG_b$
- Se mutan las probabilidades $Pj(Ij(PROG_b))$ (las instrucciones presentes en $PROG_b$)
- P_{Mp} = probabilidad de mutación
- z = número de posibles instrucciones
- Dividir por $|PROG_b|$ evita que se mute más los árboles más grandes (aunque la raíz cuadrada da mayor número de mutaciones a dichos árboles. Justificación empírica)

$$P_{M_p} = \frac{P_M}{z \cdot \sqrt{|\mathrm{PROG}_b|}}$$

Mutación del PPT

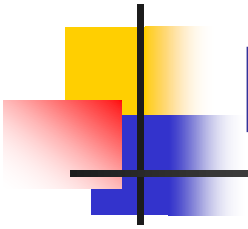
- Mutación:

$$P_j(I) := P_j(I) + mr \cdot (1 - P_j(I))$$

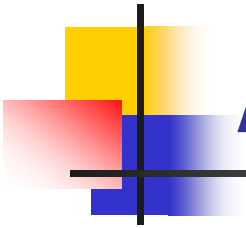
- Después hay que normalizar:

$$P_j(I) := \frac{P_j(I)}{\sum_{I^* \in S} P_j(I^*)} \qquad \forall P_j(I) : I \in S$$

Resultados PIPE

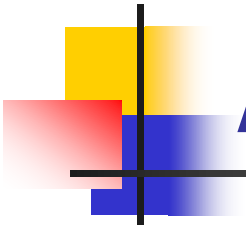


- Regresión simbólica: PIPE mejor que GP en el 24% de las ejecuciones y peor en el 33%. Mayor varianza
- 6-bit parity problem:
  - Más ejecuciones exitosas (70% vs. 60%)
  - Más rápido (52476 vs. 120000 evaluaciones)
  - Más pequeños (61 vs. 90 nodos)
- R. P. Salustowicz, M. A. Wiering, J. Schmidhuber. 1998. Learning Team Strategies: Soccer Case Studies. Machine Learning



Aplicación de PIPE a soccer

- Dominio similar a Robosoccer
- Acciones:
  - Simples: go_forward, turn_to_ball, turn_to_goal, shoot
  - Complejas: goto_ball, goto_goal, goto_own_goal, goto_player, goto_opponent, pass_to_player, shoot_to_goal



Aplicación de PIPE a soccer

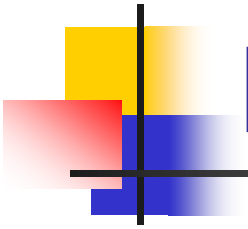
- BRO: Biased Random Opponent. Oponente casi aleatorio, pero que tiende a marcar goles al contrario (marca 75 goles a un oponente estático)
- GO: Good Opponent. Marca 417 goles a BRO
- PIPE: la fitness se calcula contra BRO
- COPIPE: la fitness se calcula con co-evolución
- TD-Q: es una especie de aprendizaje por refuerzo
- Se aprende un PPT para cada acción

Resultados PIPE fútbol contra BRO (acciones simples)

Team size		*GO*	PIPE	CO-PIPE	TD-Q
	Maximum score difference	417	310	192	42
1	Average goals ± st.d.	417 ± 6	320 ± 42	212 ± 97	52 ± 14
	Average *BRO* goals ± st.d.	0 ± 0	10 ± 7	20 ± 10	10 ± 3
	Achieved after games	n.a.	3300	3000	1700
	Maximum score difference	481	359	310	70
3	Average goals ± st.d.	481 ± 8	373 ± 86	324 ± 62	102 ± 14
	Average *BRO* goals ± st.d.	0 ± 1	14 ± 6	14 ± 11	32 ± 8
	Achieved after games	n.a.	3300	3200	1700
	Maximum score difference	364	481	357	154
11	Average goals ± st.d.	367 ± 18	512 ± 129	393 ± 53	212 ± 84
	Average *BRO* goals ± st.d.	3 ± 1	31 ± 23	36 ± 27	58 ± 23
	Achieved after games	n.a.	3100	1900	2500

Resultados PIPE fútbol contra BRO (acciones complejas)

	GO	PIPE	CO-PIPE	TD-Q
Maximum score difference	364	530	536	46
Average goals ± st.d.	367±18	551±215	539±220	76±140
Average *BRO* goals ± st.d.	3±1	21±35	3±4	30±29
Achieved after games	n.a.	1200	1200	900

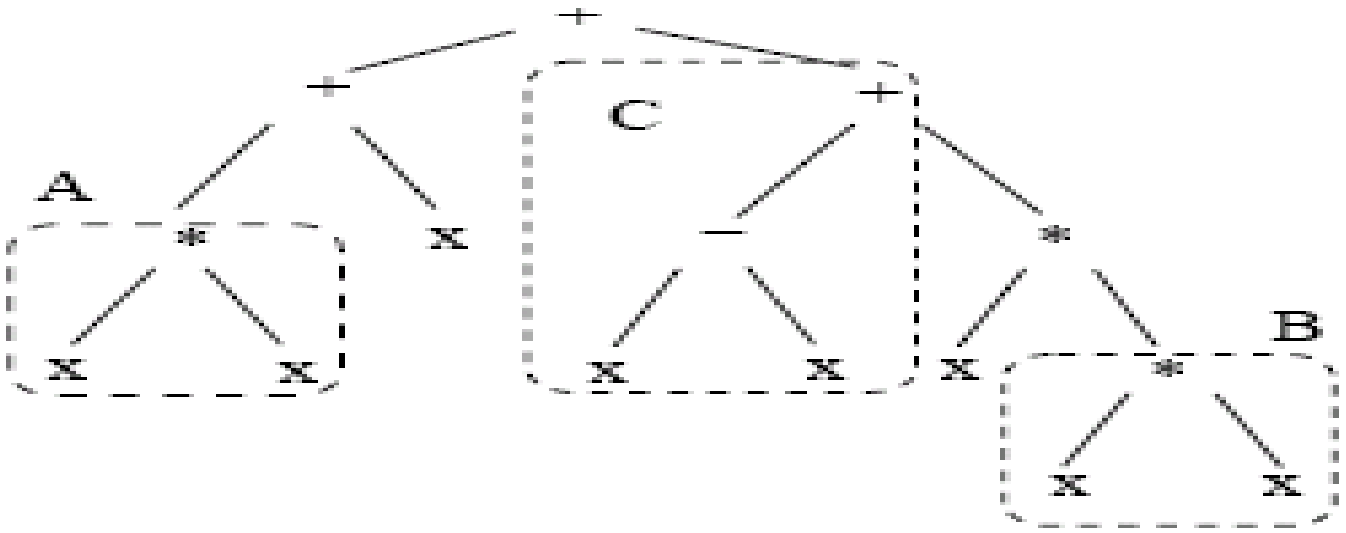


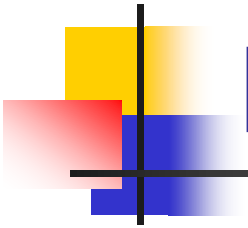
Limitaciones PIPE

- La distribución de probabilidad de cada nodo es independiente de la de los demás
- Los buenos subárboles (building blocks) son dependientes de la posición en el árbol, no pueden ser movidos a otros lugares

Limitaciones. Building blocks en distintas posiciones

$$f(x) = x^3 + x^2 + x.$$

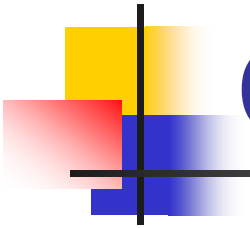




Hierarchical PIPE

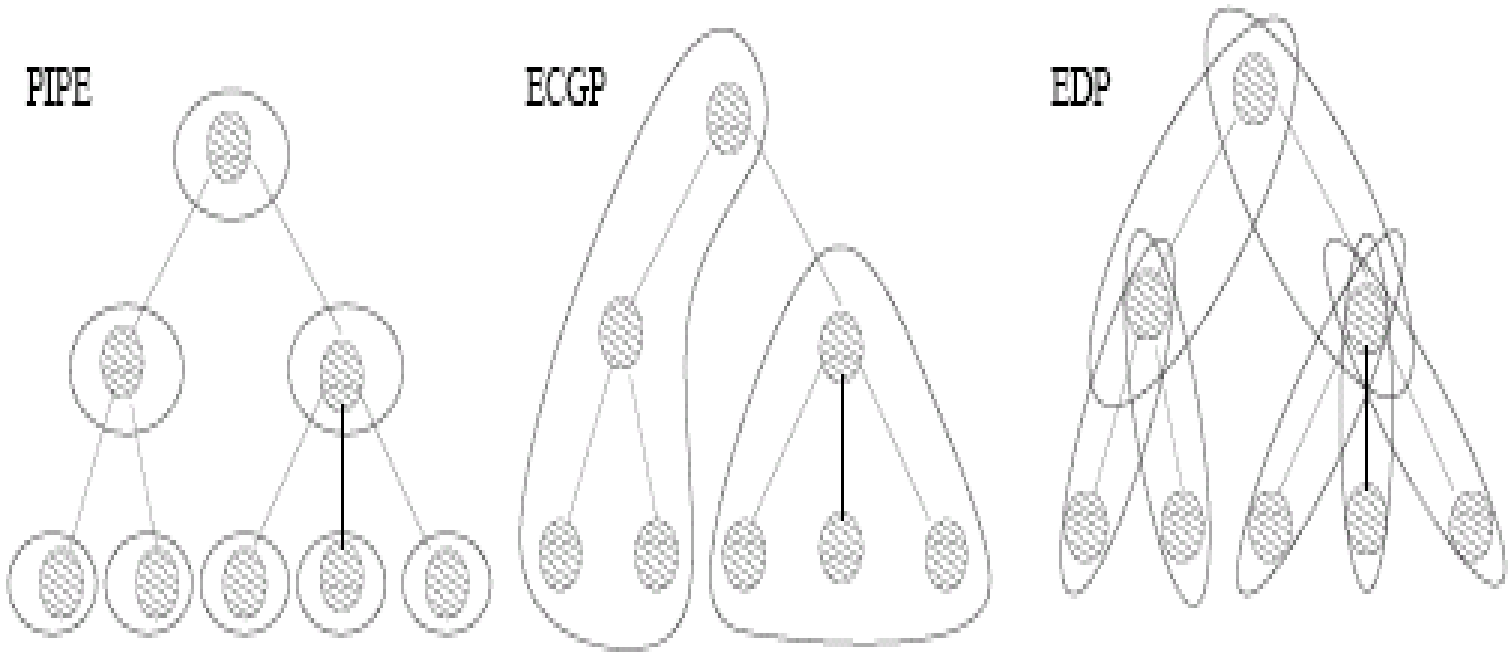
- Rafal Salustowicz, Jurgen Schmidhuber. 1998. “Evolving Structured Programs with Hierarchical Instructions and Skip Nodes”.
- Usa:
  - Jerarquía de instrucciones: las funciones de menor nivel sólo pueden ser argumentos de las de más nivel (ej: +,- arriba; sin, cos, sqrt en medio; constantes y variables abajo)
  - Skip nodes: funciones con n argumentos que sólo devuelven uno de ellos. Sirven para mantener intrones (guardan código que no se ejecuta y protegen otras instrucciones)

Hierarchical PIPE. Conclusiones

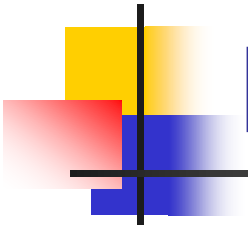


- Experimentos en regresión simbólica y multiplexor
- La jerarquía se elige a mano y esa elección es bastante responsable de los buenos resultados
- Pero muestran que sólo la combinación de la jerarquía con los *skip nodes* mejorar significativamente los resultados
- Los *skip nodes* sólo funcionan con código restringido por la jerarquía. Si no, no tienen efectos

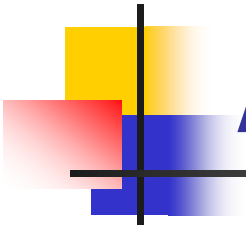
PIPE, ECJP, EDP



Estimation of Distribution Programming (EDP)



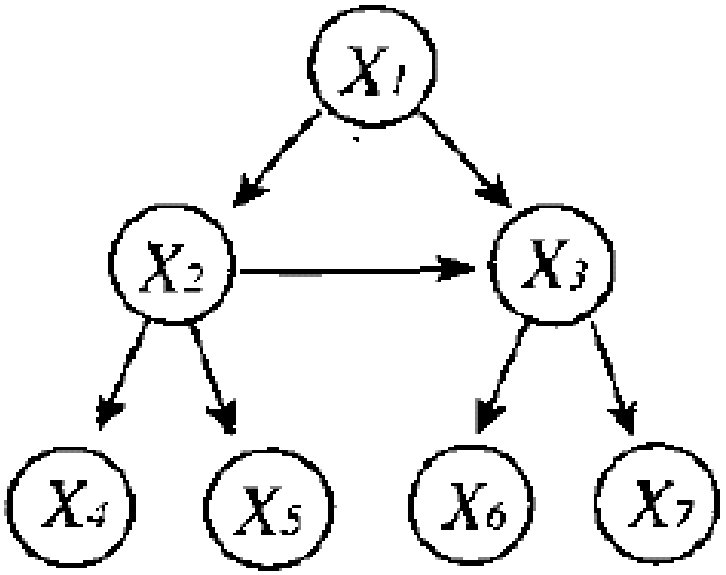
- Yanai, Iba. 2003. Estimation of distribution programming based on Bayesian network. CEC 2003.
- Utiliza probabilidad conjunta de nodo padre e hijo



Algoritmo EDP

1. Inicializar población
2. Evaluar individuos
3. Estimar la distribución
4. Si hay que terminar, ve a 7
5. Generar una nueva población
6. Reemplazar la población
7. Devolver el mejor individuo

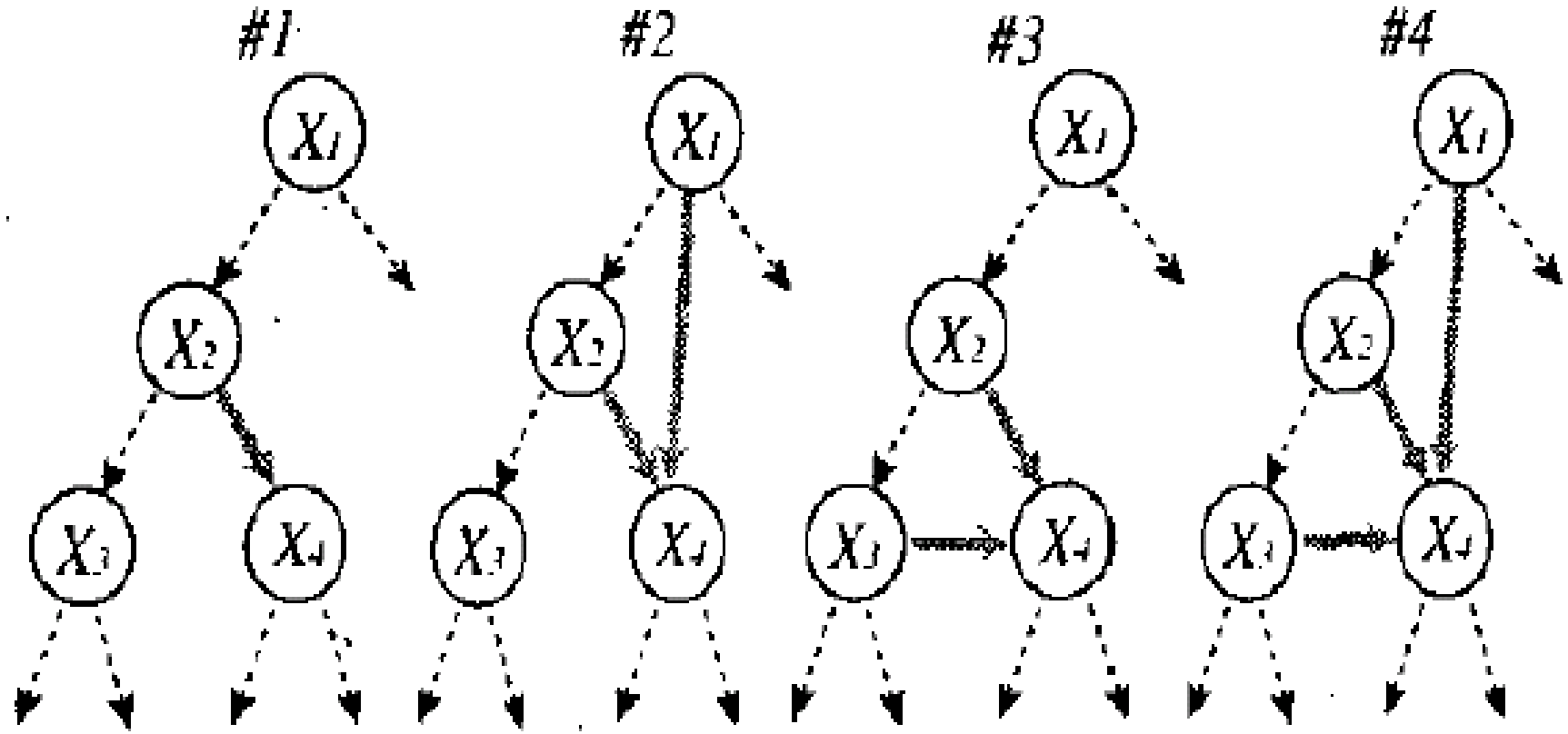
Red Bayesiana



$$P(X_3 = x_3 | X_1 = x_1, X_2 = x_2, \cdots, X_6 = x_6, X_7 = x_7,)$$
$$= P(X_3 = x_3 | X_1 = x_1, X_2 = x_2). \qquad (1)$$

Fuente: Yanay e Iba, 2003

Posibles Redes Bayesianas



Fuente: Yanay e Iba, 2003

Tamaños de las tablas condicionales

- M = posibles símbolos
- N = nodos en el árbol
- I = dependencias
- La tabla tendrá M*M*…*M = M^I líneas
- Cada línea tendrá M probabilidades

Padre, hermano	+	-	*	/
+,+	0.3	0.3	0.2	0.2
+,-	0.6	0.2	0.1	0.1
+,*	0.1	0.2	0.3	0.4
+,/	0.5	0.1	0.1	0.3
…				

Cálculo de probabilidades

$$P(X_i = x|C_i = c) = \frac{\sum_{j=1}^{N} \delta(j, X_i = x|C_i = c)}{N},$$

where

$$\delta(j, X_i = x|C_i = c) = \begin{cases} 1 & \text{if } X_i = x \text{ and } C_i = c \\ & \text{at the individual } j \\ 0 & \text{else} \end{cases}$$

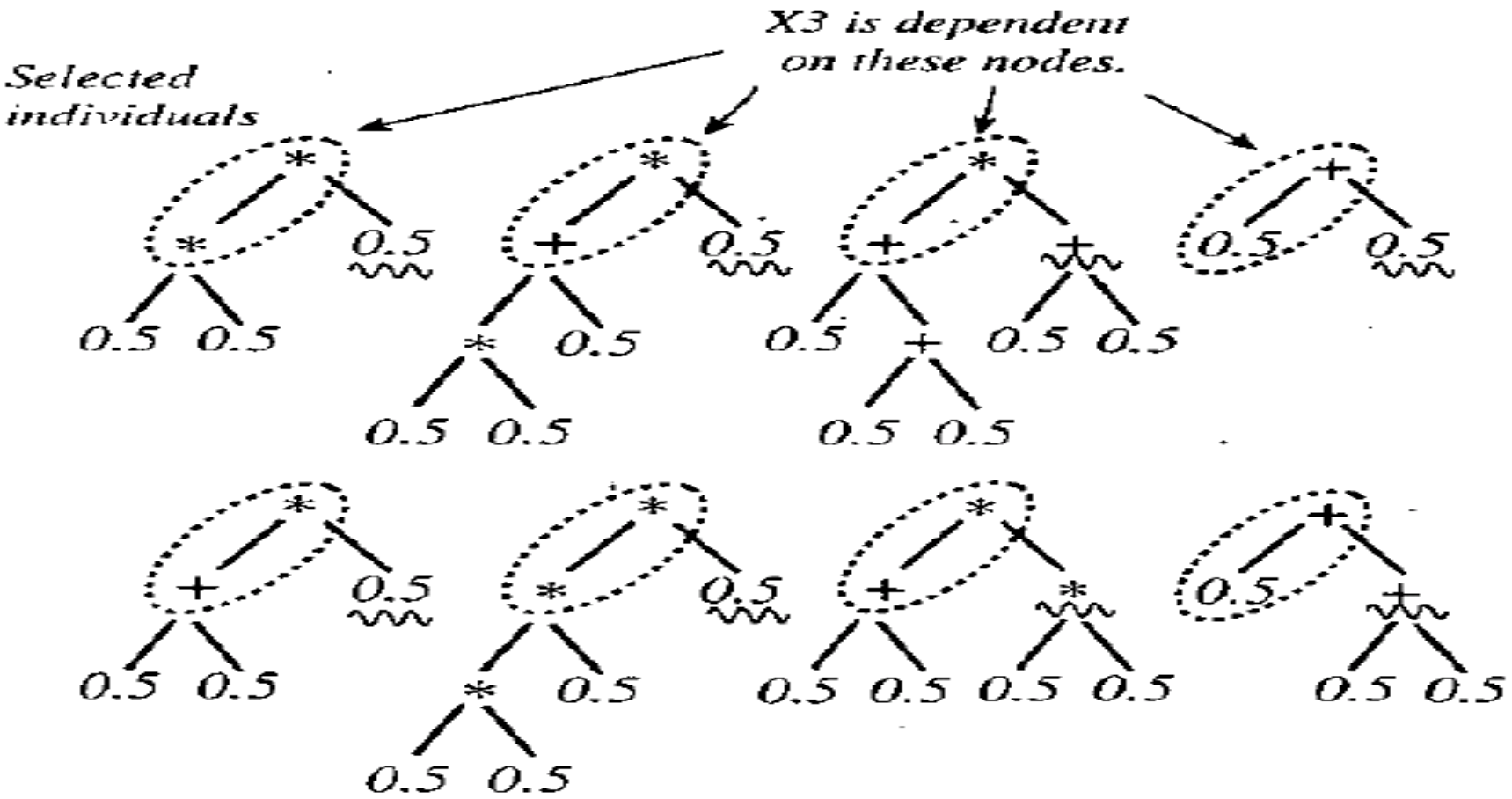
j: individuo j

Cálculo de probabilidades ponderadas por fitness

$$P(X_i = x | C_i = c) = \frac{\sum_{j=1}^{N} F_j \delta(j, X_i = x | C_i = c)}{\sum_{j=1}^{N} F_i}$$

La distribución de probabilidad se genera a partir de la población, sin considerar la distribución anterior (a diferencia de PIPE)

Individuos seleccionados



Nota: **sólo** se consideran los nodos padre y hermano

Fuente: Yanay e Iba, 2003

Distribución a partir de los individuos seleccionados

Table 2: Estimating $P(X_3 = x_3 | X_1 = x_1, X_2 = x_2)$

X_3	X_1	X_2	Frequency	Probability (= P)	Modified probability (= P')
+	+	0.5	1	0.5	0.49
*	+	0.5	0	0	0.02
0.5	+	0.5	1	0.5	0.49
+	*	+	1	0.25	0.26
*	*	+	1	0.25	0.26
0.5	*	+	2	0.5	0.48
+	*	*	0	0	0.03
*	*	*	0	0	0.03
0.5	*	*	**2**	**1.0**	**0.94**

Fuente: Yanay e Iba, 2003

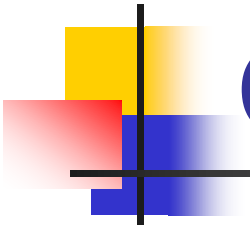
Corrección de la distribución

$$P' = (1-\alpha)P(X_i = x|C_i = c) + \alpha P_{default}(X_i = x|C_i = c).$$

$$P_{default}(X_i = x|C_i = c) = \frac{1}{node\ symbol\ number}.$$

Es una especie de probabilidad a priori. Así, todas las probs. son distintas de cero

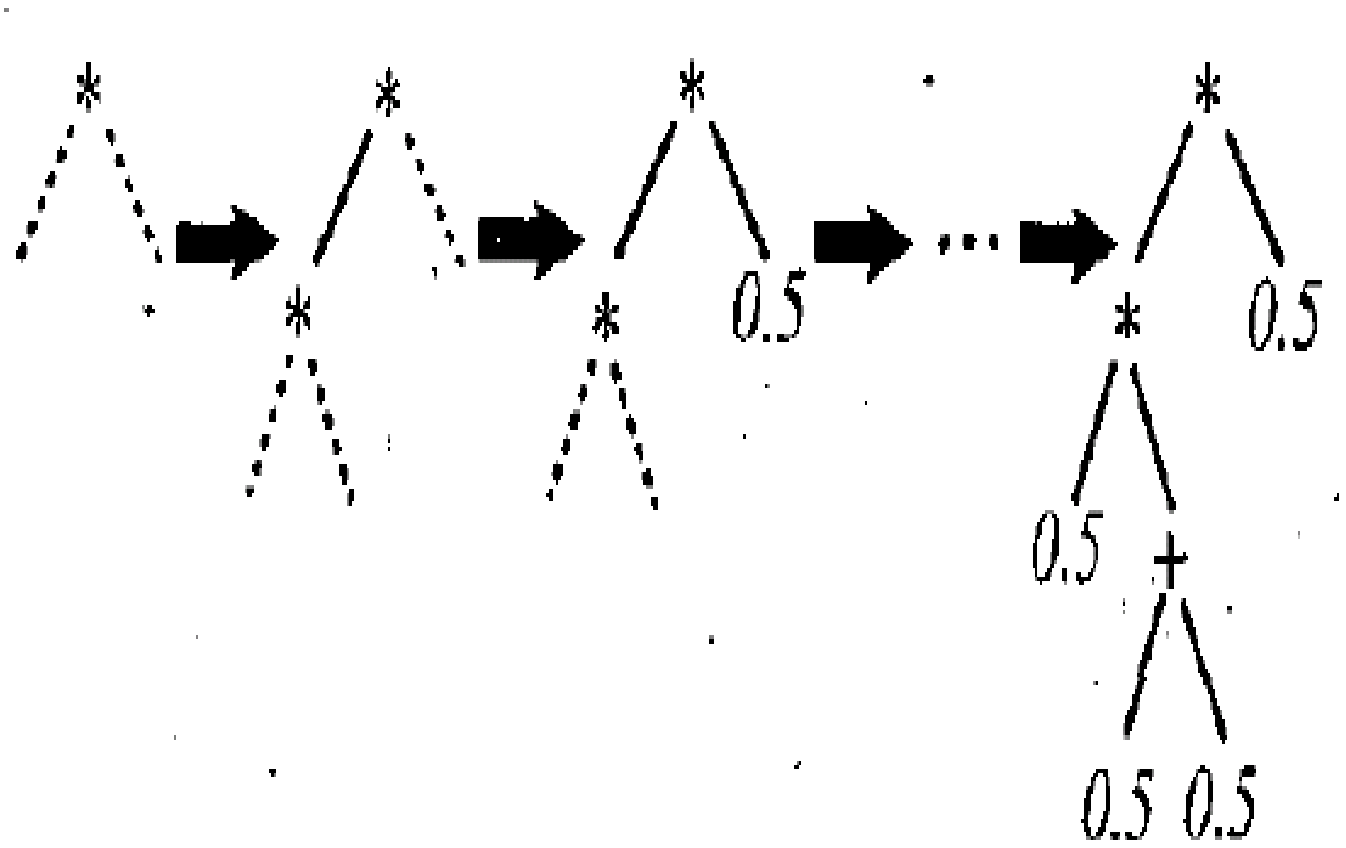
(número de símbolos)



Generación de programas

- Primero se selecciona los mejores k individuos de la población anterior

Generación de individuos



Fuente: Yanay e Iba, 2003

El "Max Problem"

Conseguir el valor máximo con +, * y 0.5

Fácil para EDP-GP porque 0.5 debe aparecer abajo, + en medio y * arriba

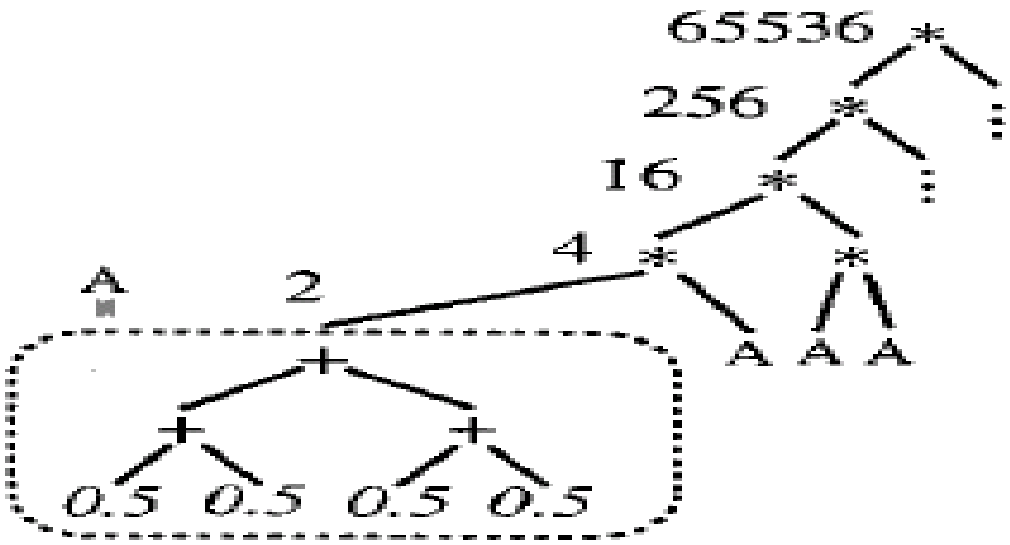
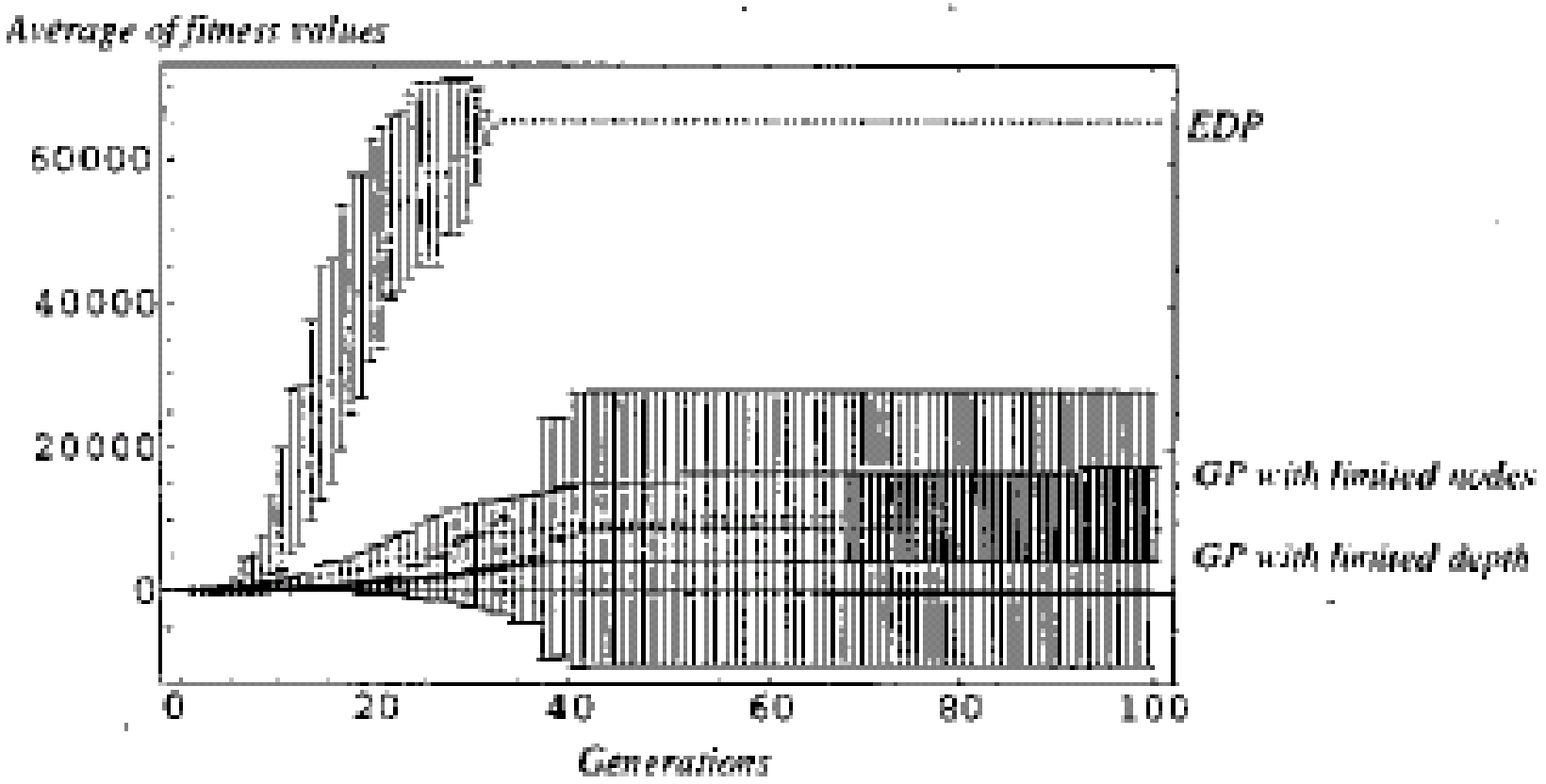


Figure 6: The maximum value by a limited depth tree.

Fuente: Yanay e Iba, 2003

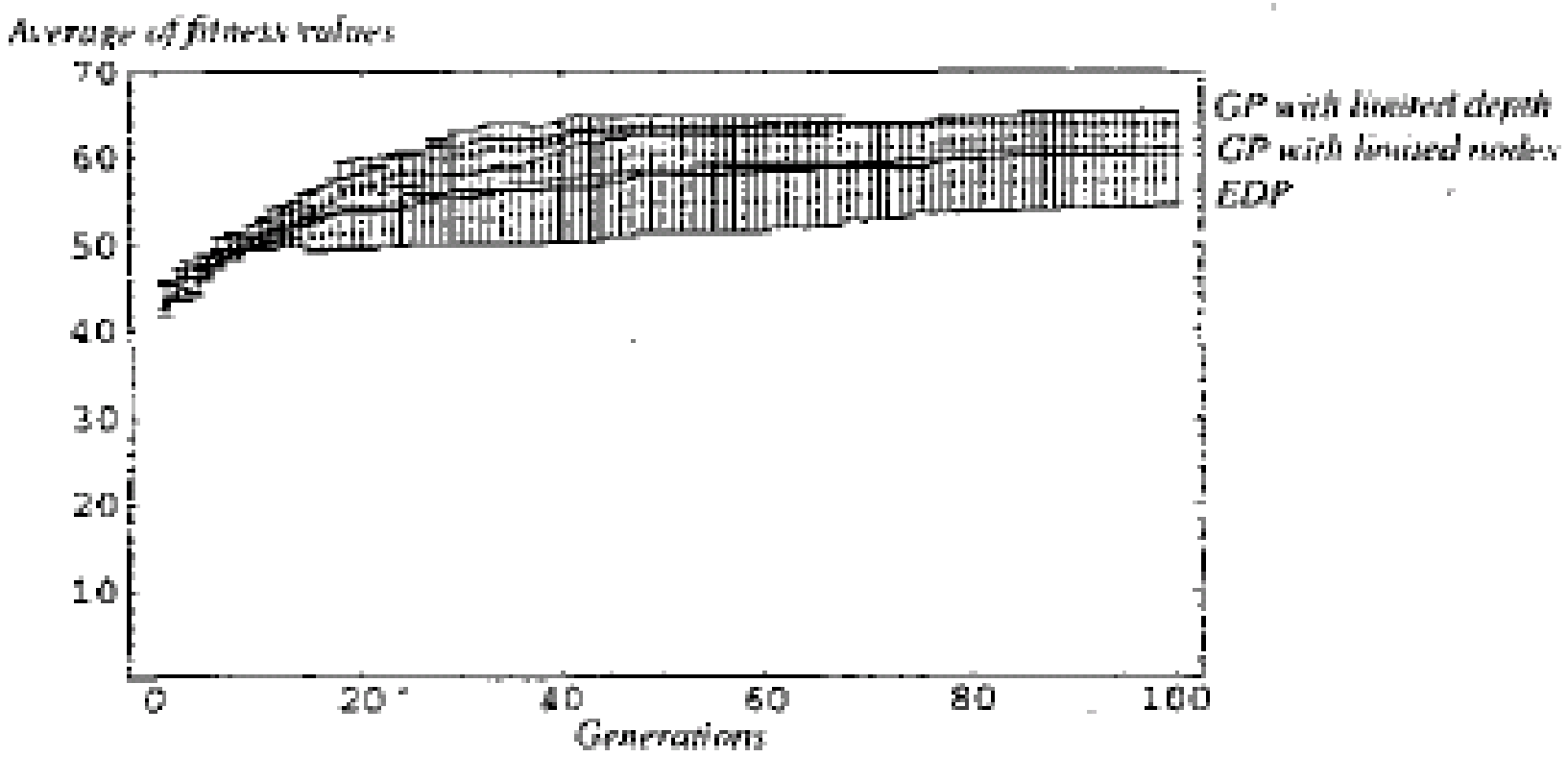
Resultados. Max Problem



Fuente: Yanay e Iba, 2003

Resultados. 6-Multiplexer

Aquí EDP funciona algo peor



Fuente: Yanay e Iba, 2003

Redes Bayesianas Recursivas

La red se va encajando recursivamente en distintas partes del árbol. Disminuye el tamaño de la red y hace que las dependencias puedan ocurrir en cualquier parte del árbol

Figure 10: The method used in experiments

Figure 11: Recursive single Bayesian network

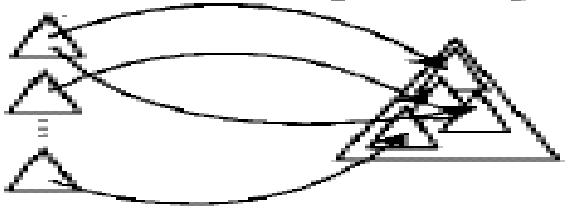
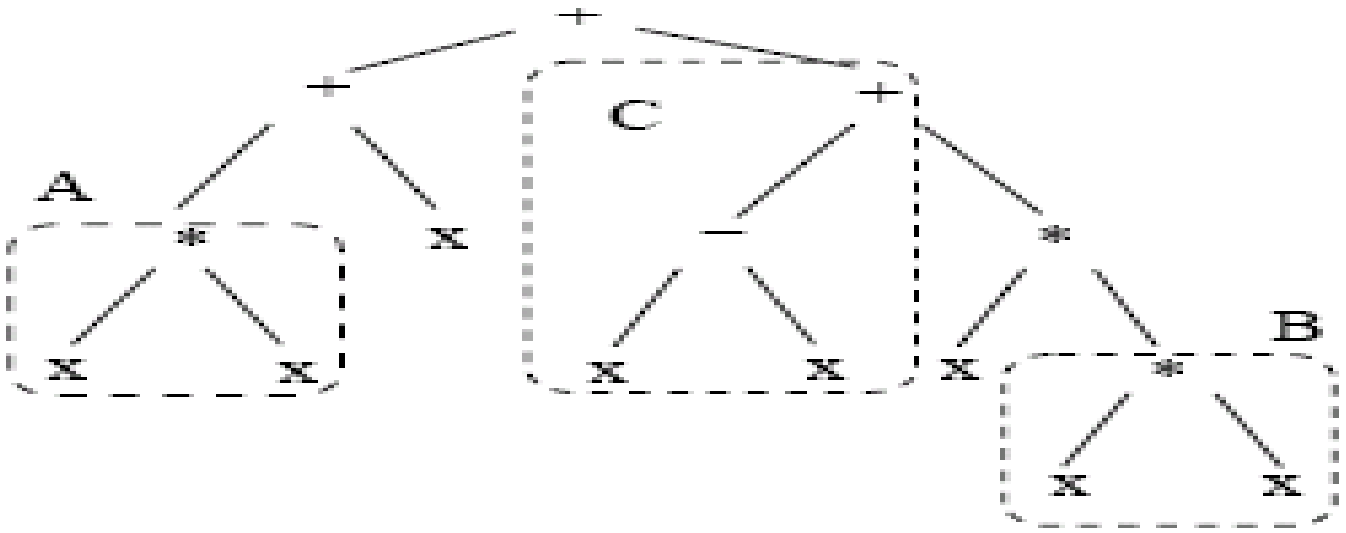


Figure 12: Recursive multi-Bayesian network

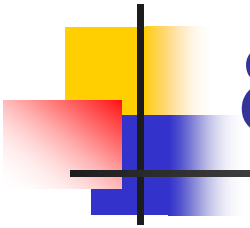
Fuente: Yanay e Iba, 2003

Limitaciones. Building blocks en distintas posiciones

$$f(x) = x^3 + x^2 + x.$$



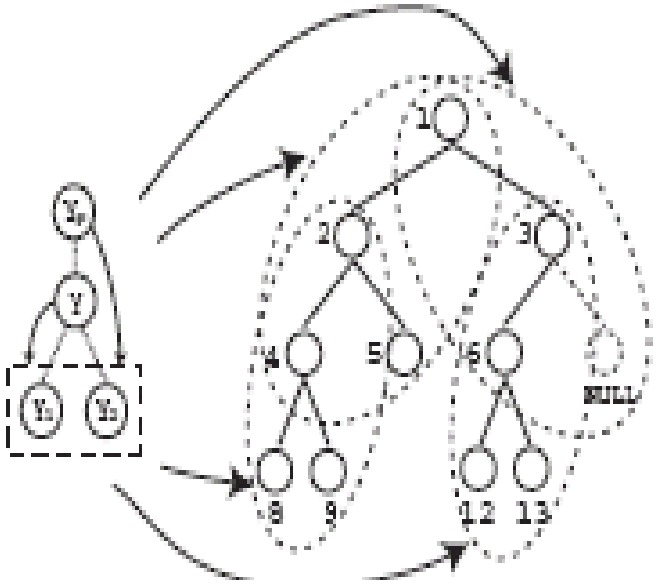
Extended EDP (XEDP) [Yanai & Iba, 05]



- Usa un modelo de probabilidad condicional "de posición absoluta" de EDP. Aprende la estructura
- Y un modelo de probabilidad condicional "de posición relativa" (redes recursivas)

XEDP. Redes bayesianas

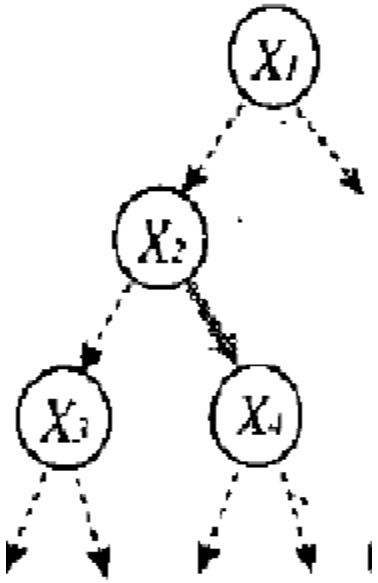
Dist. Recursiva (relativa)



$$P(Y_1, Y_2, \cdots, Y_{a_{max}} | Y, Y_p)$$

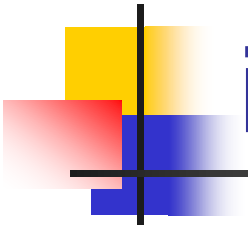
P(Hijos|padre,abuelo)

Dist. Condicional (absoluta)



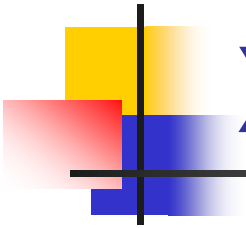
P(hijo|padre)

XEDP. Algoritmo para generar individuos



- Combina las dos distribuciones (absoluta y relativa)

1. Generar un programa T utilizando la distribución absoluta
2. Generar un subárbol S utilizando la distribución relativa
3. Reemplazar un subárbol cualquiera de T por S



XEDP. Experimentos

- XEDP
- GP
- Tipo A: usa solo la distribución condicional (absoluta)
- Tipo B: usa sólo la distribución recursiva (relativa)
- Tipo C: XEDP, pero reemplazando el subárbol por otro aleatorio
- Tipo D: XEDP, pero la distribución absoluta es como en PIPE (sin dependencias entre padre e hijo)

Resultados Max Problem

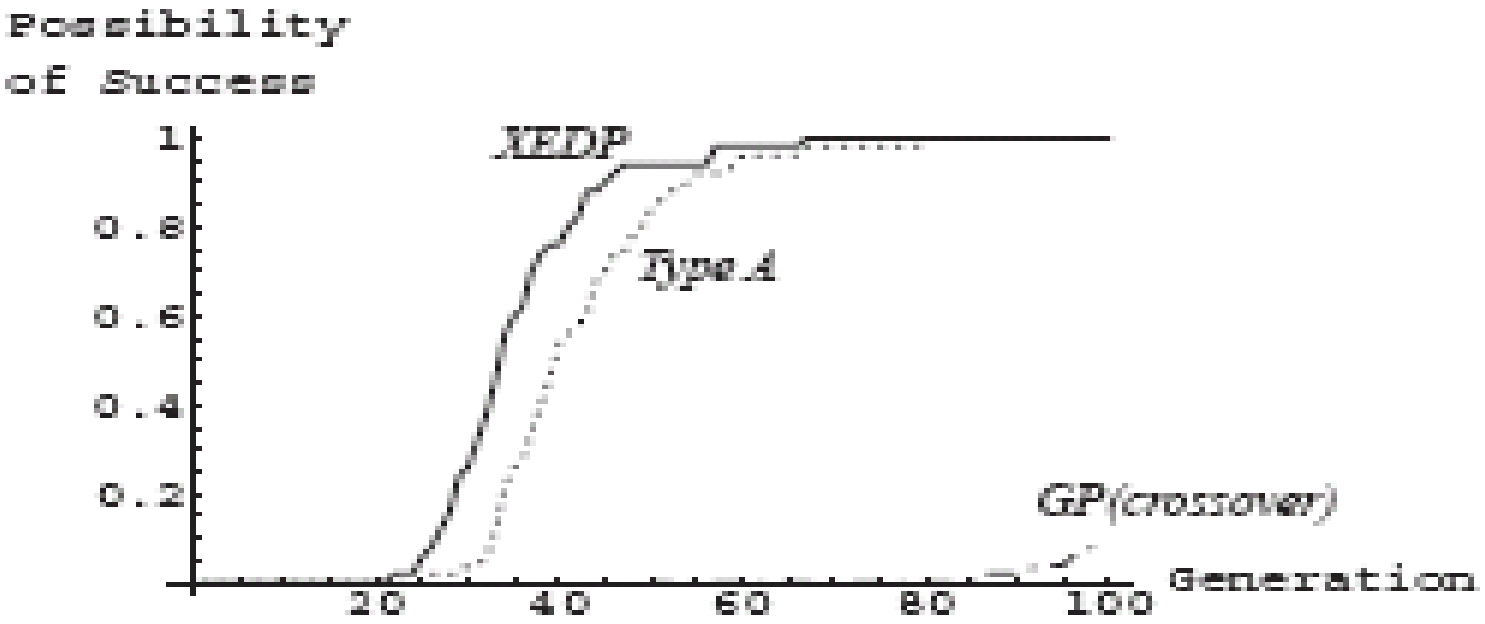


Figure 2: Cumulative probability of success for the Max problem.

Resultados 6-multiplexer

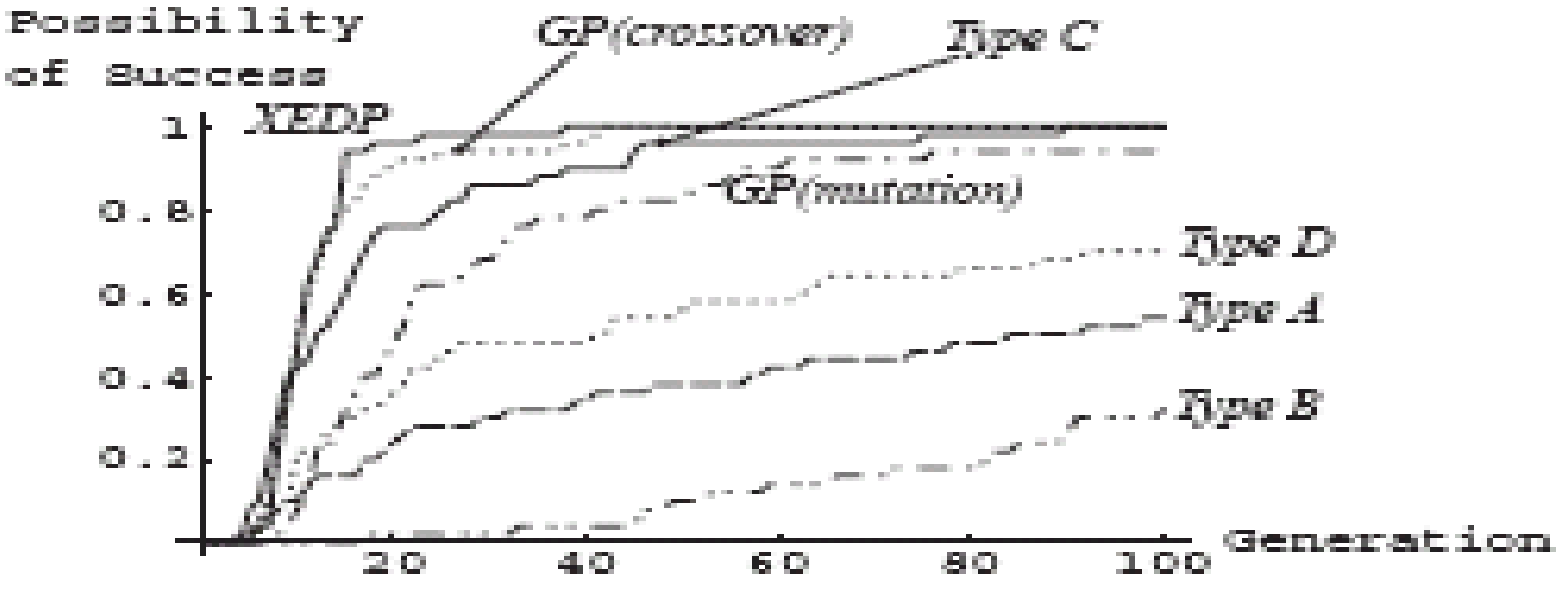


Figure 3: Cumulative probability of success for the Boolean 6-bit multiplexer problem.

Resultados Wall-following

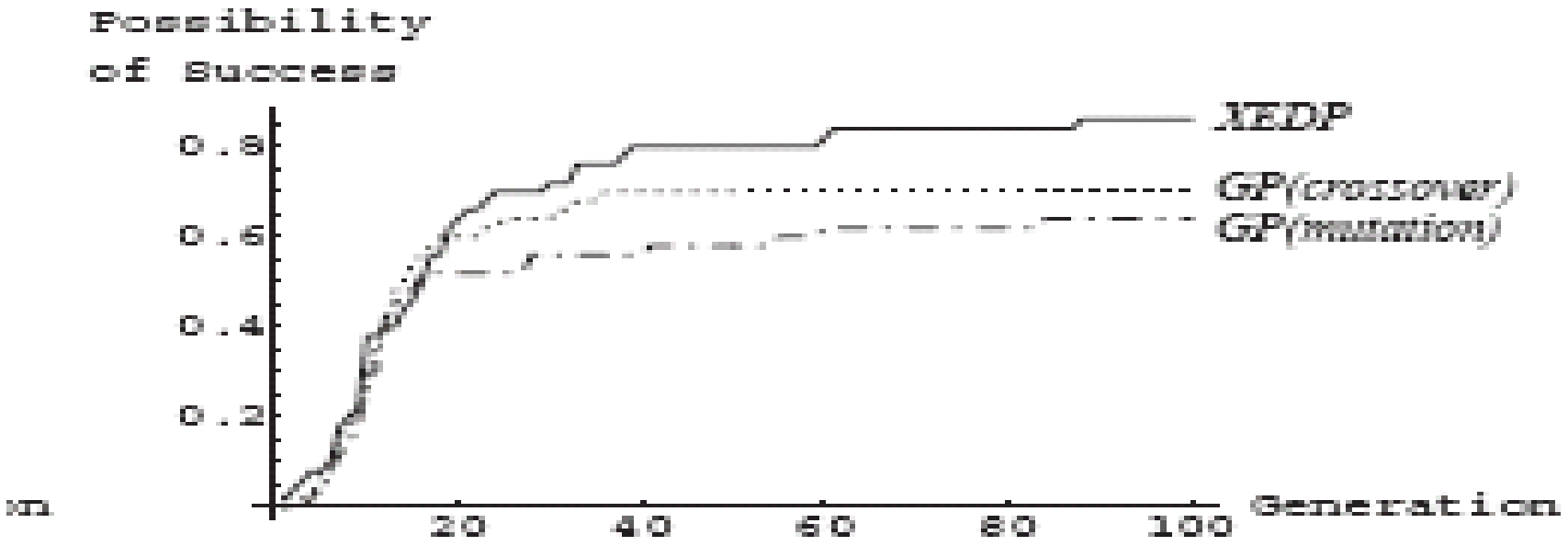
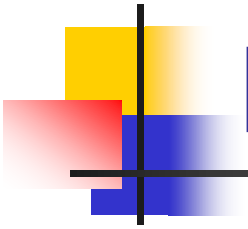


Figure 4: Cumulative probability of success for the Wall-following problem.

Extended Compact Genetic Programming (ECGP)

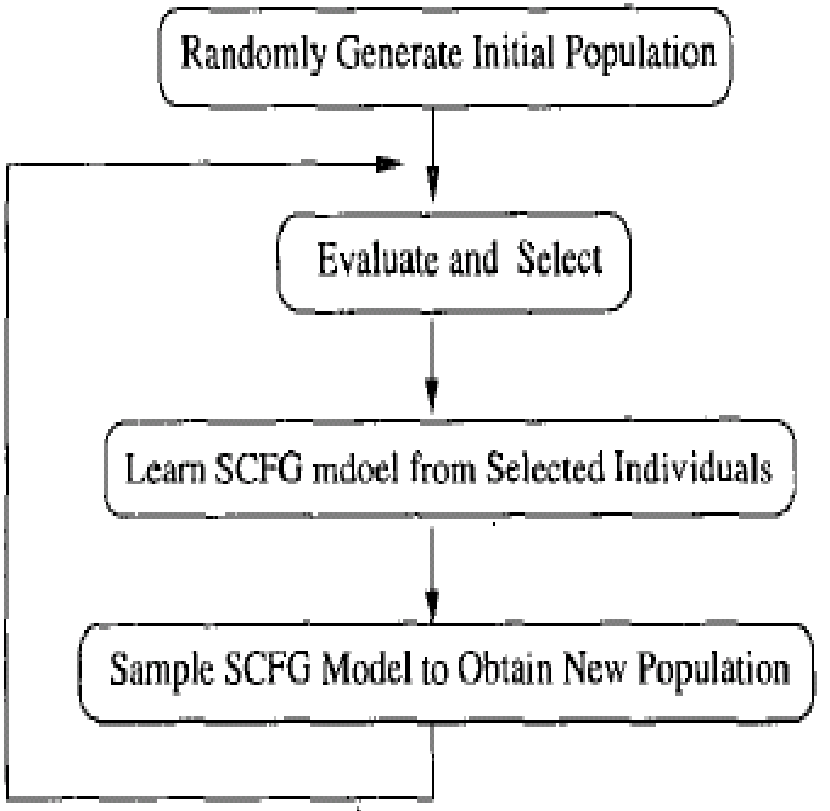
- K. Sastry, D.E. Goldberg. 2003. Probabilistic model building and competent genetic programming. Genetic Programming Theory and Practice
- Utiliza Marginal Product Models
- Parte el PPT en varios subárboles, supuestamente independientes
- Cada subárbol con probabilidades conjuntas (no asume independencia de los nodos)



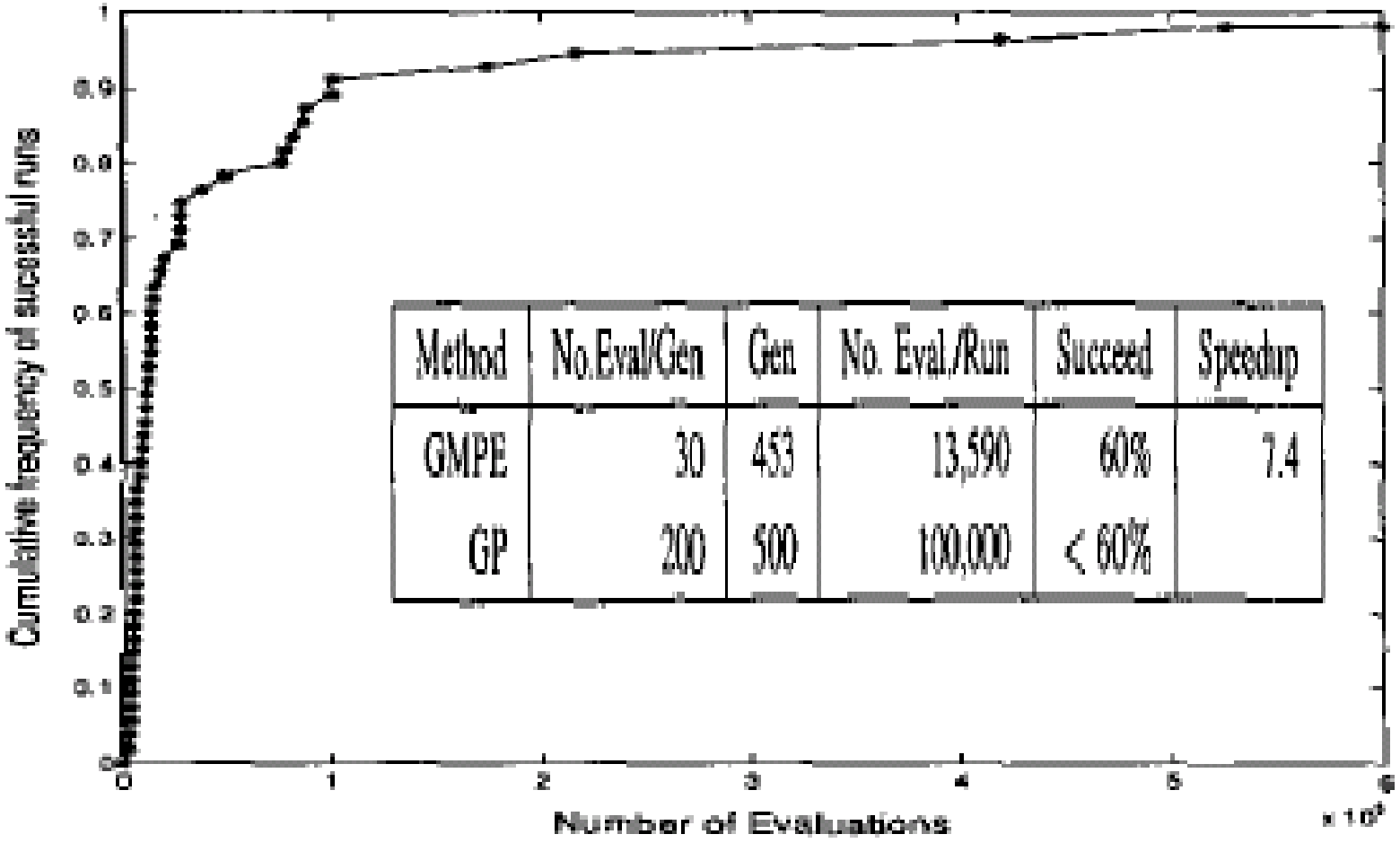
EDA's con gramáticas

- Shan, McKay, Baxter, Abbass, Essam, Nguyen. 2004. Grammar Model-Based Program Evolution: **GMPE**
- Aprender gramáticas de contexto libre estocásticas (SCFG)
- Estocática: cada regla de reescritura tiene un peso que indica la probabilidad con la que se la tiene que usar

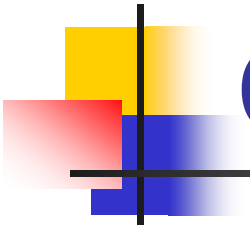
Algoritmo GMPE



Resultados. Max Problem



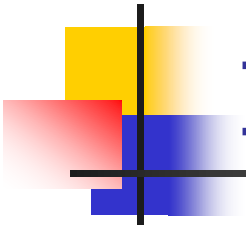
Method	No.Eval/Gen	Gen	No. Eval./Run	Succeed	Speedup
GMPE	30	453	13,590	60%	7.4
GP	200	500	100,000	< 60%	



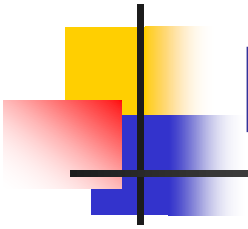
Conclusiones EDA-GP

- Los algoritmos EDA-GP exploran las distribuciones de probabilidad que generan programas con el objetivo de encontrar una que genere la solución
- Usan distribuciones de probabilidad de manera explícita
- También pueden aprender gramáticas estocásticas
- Menos probados, pero parece que los resultados son equivalentes o mejores a PG

Inductive Functional Programming

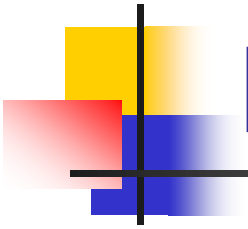


- R. Olsson. **Inductive Functional Programming Using Incremental Program Transformation**. *Artificial Intelligence Journal.* 74:1. 1995
- ADATE: Automatic Design of Algorithms Through Evolution
- La PG no suele evolucionar recursividad o bucles
- El cruce es un mal operador de transformación de programas



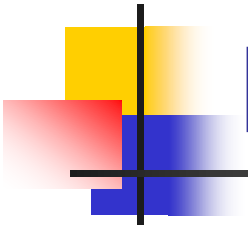
Especificaciones ADATE

- Un conjunto de **tipos**
- Funciones primitivas
- Tipo del resultado
- Un conjunto de entradas. Entradas de dificultad incremental y con casos especiales
- Una función (fitness) que evalúa la corrección del algoritmo con las entradas/salidas



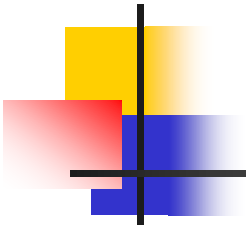
Búsqueda en ADATE

- Función heurística (fitness):
  - Corrección
  - Complejidad sintáctica (tamaño)
  - Complejidad en tiempo
- Iterative deepening (IDA*)



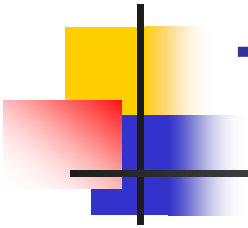
Lenguaje en ADATE

- Subconjunto de ML (lenguaje funcional)
- Definición de tipos
- Definición de funciones
- Let
- Case



Búsqueda en ADATE

- Se parte del programa vacío
- Se progresa explorando todos los programas de menor a mayor tamaño (navaja de Occam)
- Heurísticas en generación de expresiones:
  - En sentencias case, se debe activar más de una rama
  - En llamadas recursivas debe haber un parámetro que se vaya decrementando y un caso base
- Se generan nuevos programas mediante transformaciones compuestas (secuencias de trans. atómicas)
- Iterative Deepening (IDA*)



Transformaciones atómicas

- Reemplazo (R)
- Reemplazo que no empeora el programa (REQ)
- Abstracción: invención de nuevas funciones (ABSTR)
- Distribución case (CASE-DIST)
- “Embedding”: cambiar el tipo de la función para hacerla más general (EMB)

Ejemplo de reemplazo

```
fun sort Xs = case Xs of nil => Xs | X1::Xs1 => ?
```

```
fun sort Xs =
  case Xs of nil => Xs
  | X1::Xs1 => case Xs1 of nil => Xs | X2::Xs2 => ?
```

```
fun sort Xs =
  case Xs of nil => Xs
  | X1::Xs1 =>
  case Xs1 of nil => Xs
  | X2::Xs2 => case X2<X1 of true => ? | false => Xs
```

```
fun sort Xs =
  case Xs of nil => Xs
  | X1::Xs1 =>
  case sort Xs1 of nil => Xs
  | X2::Xs2 => case X2<X1 of true => ? | false => Xs
```

Ejemplo de abstracción

```
case sort Xs1 of nil => Xs
| X2::Xs2 => case X2<X1 of true => ? | false => Xs
```

```
fun sort Xs =
  case Xs of nil => Xs
  | X1::Xs1 =>
  let fun g V1 =
    case V1 of nil => Xs
    | X2::Xs2 => case X2<X1 of true => ? | false => Xs
  in
    g(sort Xs1)
  end
```

Distribución case

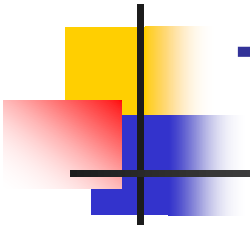
$h(A_1, \ldots, A_i,$ `case` E `of` $Match_1$ `=>` E_1 | ... | $Match_n$ `=>` $E_n, A_{i+1}, \ldots, A_m)$

`case` E `of`

$$Match_1 \Rightarrow h(A_1, \ldots, A_i, E_1, A_{i+1}, \ldots, A_m)$$

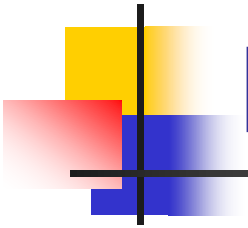
$$\vdots$$

$$\mid Match_n \Rightarrow h(A_1, \ldots, A_i, E_n, A_{i+1}, \ldots, A_m)$$



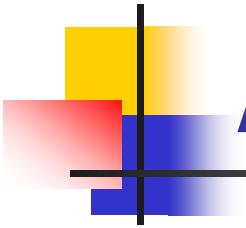
Transformaciones compuestas

1. REQ $\Rightarrow$ R. The R is applied in the expression introduced by the REQ.
2. REQ $\Rightarrow$ ABSTR. The ABSTR is such that the expression introduced by the REQ occurs in the $H(E_1, \ldots, E_n)$ used by the ABSTR but not entirely in H.
3. ABSTR $\rightarrow$ R. The R is applied in the the right hand side $H(V_1, \ldots, V_n)$ of the `let`-definition introduced by the ABSTR.
4. (a) ABSTR $\rightarrow$ REQ! or (b) ABSTR $\rightarrow$ REQ! REQ!. The REQ(s) are applied in $H(V_1, \ldots, V_n)$.
5. ABSTR $\Rightarrow$ EMB!. The `let`-function introduced by the ABSTR is embedded.
6. CASE-DIST $\Rightarrow$ ABSTR. The ABSTR is such that the root of $H(E_1, \ldots, E_n)$ was marked by the CASE-DIST.
7. CASE-DIST $\Rightarrow$ R. The R is such that the root of the expression *Sub*, which is replaced by the R, was marked by the CASE-DIST.
8. EMB $\rightarrow$ R. The R is applied in the right hand side of the definition of the embedded function.



Problemas resueltos

- Simplifica polinomios: si hay coeficientes repetidos del mismo grado, los suma
- Intersección de rectángulos: devuelve la intersección de dos rectángulos, si la hay
- Generación de permutaciones: genera todas las permutaciones de una lista
- Container: mover cajas pequeñas dentro de container (http://www-ia.hiof.no/~geirvatt/)
- Otros: transposición de matrices, búsqueda de subcadenas, multiplica dos números binarios, encuentra caminos en grafos, ...



ADATE. Resultados

- Simplificación de polinomios
  - ["+", "=", "false", "true","term", "nil", "cons"]
- Intersección de rectángulos
  - ["<", "point", "rect", "none", "some"]
- Inserción/borrado en árboles binarios
  - ["<", "bt_nil", "bt_cons", "false", "true"]
- Inversión, intersección, borrado en listas
  - ["false", "true", "=", "nil", "cons"]
- Generación de permutaciones
  - ["false", "true", "nil"", "cons", "append"]
- Ordenación
  - ["false", "true", "<", "nil", "cons"]

ADATE. Resultados

Problem	*Run time in days:hours*
Polynomial simplification	0:7
Rectangle intersection	1:18
BST deletion	7:12
BST insertion	3:5
List reversal	0:10
List intersection	6:3
List delete min	8:8
Permutation generation	9:5
List sorting	1:12
List splitting	0:7

ADATE. Sort

```
program =
fun f (V2_4) =
  case V2_4 of
    nil => V2_4
  | cons( V2_aa, V2_ab ) =>
  let
    fun g2_d84e8 (V2_d84e9) =
      case V2_d84e9 of
        nil => cons( V2_aa, nil )
      | cons( V2_cbe81, V2_cbe82 ) =>
      case (V2_aa < V2_cbe81) of
        false => cons( V2_cbe81, g2_d84e8( V2_cbe82 ) )
      | true =>
      cons( V2_aa, V2_d84e9 )
  in
    g2_d84e8( f( V2_ab ) )
  end
```

ADATE. Intersección de rectángulos

```
fun f
      ((
         ( V2_5 as rect(
                     ( V2_6 as point( V2_7, V2_8 ) ),
                     ( V2_9 as point( V2_a, V2_b ) )
                     ) ),
         ( V2_c as rect(
                     ( V2_d as point( V2_e, V2_f ) ),
                     ( V2_10 as point( V2_11, V2_12 ) )
                     ) )
         )) =
  case (V2_a < V2_e) of
    false => (
      case (V2_7 < V2_11) of
        false => none
      | true =>
      case (V2_8 < V2_12) of
        false => none
      | true =>
      case (V2_b < V2_f) of
        false => some(
                   rect(
                     point(
                       case (V2_e < V2_7) of false => V2_e | true => V2_7,
                       case (V2_8 < V2_f) of false => V2_8 | true => V2_f
                       ),
                     point(
                       case (V2_a < V2_11) of false => V2_11 | true => V2_a,
                       case (V2_b < V2_12) of false => V2_12 | true => V2_b
                       )
                     )
                   )
      | true =>
      none
      )
```