# Introducción a la Computación Evolutiva

Carlos A. Coello Coello

*carlos.coellocoello@ccinvestav.mx*
CINVESTAV-IPN
Evolutionary Computation Group (EVOCINV)
Departamento de Computación
Av. IPN No. 2508, Col. San Pedro Zacatenco
México, D.F. 07360, MEXICO

**Clase 8**

## Cruza Aritmética

Es una variante de la cruza intermedia, en la cual se toman en cuenta los límites inferiores y superiores de las variables del problema.

La única diferencia con respecto a la cruza intermedia es el cálculo de "**a**":

$$a \in= \begin{cases} |max(\alpha,\beta), min(\gamma,\delta)| & \text{si } v_k > w_k \\ |0,0| & \text{si } v_k = w_k \\ |max(\gamma,\delta), min(\alpha,\beta)| & \text{si } v_k < w_k \end{cases} \quad (1)$$

## Cruza Aritmética

Para fines del cálculo de "**a**":

$$\alpha = \frac{l_k^w - w_k}{v_k - w_k} \qquad \beta = \frac{u_k^v - v_k}{w_k - v_k}$$

$$\gamma = \frac{l_k^v - v_k}{w_k - v_k} \qquad \delta = \frac{u_k^w - w_k}{v_k - w_k}$$

Donde cada valor $v_k$ está en el rango:
$[l_k^v, u_k^v]$ y cada valor $w_k$ está en el rango $[l_k^w, u_k^w]$

## Cruza Aritmética (Ejemplo Parte 1)

**P1** = $< 2.3,\ 4.5,\ -1.2,\ 0.8 >$
**P2** = $< 1.4,\ -0.2,\ 6.7,\ 4.8 >$

Supondremos, que para toda $v_k$ y para toda $w_k$, el rango es: [-2,7].

Si ahora ubicamos el punto de cruza entre la segunda y la tercera variable, tenemos:

**H1** =

$$< 2.3,\ 4.5,\ 6.7 * a_1\ - 1.2 * (1 - a_1),\ 4.8 * a_2\ + 0.8 * (1 - a_2) >$$

**H2** =

$$< 1.4, -0.2,\ -1.2a_1\ + 6.7 * (1 - a_1),\ 0.8 * a_2\ + 4.8 * (1 - a_2) >$$

## Cruza Aritmética (Ejemplo Parte 2)

Haciendo $k$ =3, tenemos: $v_k$ = -1.2, $w_k$ = 6.7, por lo que: $w_k > v_k$

$$\alpha = \frac{\text{-2 - 6.7}}{\text{-1.2 - 6.7}} = 1.1 \quad \beta = \frac{\text{7 + 1.2}}{\text{6.7 + 1.2}} = 1.04$$

$$\gamma = \frac{\text{-2 + 1.2}}{\text{6.7 + 1.2}} = \text{-0.101} \quad \delta = \frac{\text{7 - 6.7}}{\text{-1.2 - 6.7}} = \text{-0.038}$$

Por lo que: $a_1 \in [0.038, 1.04]$

## Cruza Aritmética (Ejemplo Parte 3)

Como los rangos para la cuarta variable son los mismos que para la tercera, los cálculos son idénticos.

Es decir: $a_2 \in [0.038, 1.04]$

Supongamos que $a_1 = 0.8$ y $a_2 = 0.9$

**H1** = $< 2.3, 4.5, 5.12, 4.4 >$
**H2** = $< 1.4, -0.2, 0.38, 1.2 >$

## Cruza Aritmética Total

Es igual que la anterior, sólo que en este caso no se tiene que elegir punto de cruza, pues se aplica la misma fórmula a todos los genes del cromosoma (o sea, a todas las variables de decisión).

Si no se pre-define un rango para cada variable, entonces se usa simplemente $a \in [0, 1]$, como en el caso de la cruza intermedia.

# Cruzas para Representación Real

## Cruza Aritmética Total (Ejemplo)

**P1** = $<$ 2.3, 4.5, -1.2, 0.8 $>$
**P2** = $<$ 1.4, -0.2, 6.7, 4.8 $>$

Usando **a** = 0.6, tenemos:
**H1** = $<$ 1.76, 1.68, 3.54, 3.2 $>$
**H2** = $<$ 1.94, 2.62, 1.96, 2.4 $>$

# Cruzas para Representación Real

## Simulated Binary Crossover (SBX)

Esta técnica de cruza fue propuesta por Deb & Agrawal [1995].

Intenta emular el efecto de la cruza de un punto usada con representación binaria.

## Algoritmo (Parte 1)

1. Generar un número aleatorio "**u**" entre 0 y 1.
2. Calcular $\vec{\beta}$.

Kalyanmoy Deb and Ram Bhushan Agrawal, "**Simulated Binary Crossover for Continuous Search Spaces**", *Complex Systems*, Vol. 9, pp. 115–148, 1995.

## Algoritmo (Parte 2)

donde:

$$\vec{\beta} = \begin{cases} (2u)^{\frac{1}{\eta_c+1}} & \text{si } u \leq 0.5 \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{\eta_c+1}} & \text{de lo contrario} \end{cases} \quad (2)$$

Los autores de la técnica sugieren: $\eta_c$ = 1 ó 2.

# Cruzas para Representación Real

## Algoritmo (Parte 3)

Producir los hijos con:

**H1** = $0.5[\ (P1 + P2) - \vec{\beta}\ |\ P2 - P1\ |\ ]$

**H2** = $0.5[\ (P1 + P2) + \vec{\beta}\ |\ P2 - P1\ |\ ]$

# Cruzas para Representación Real

## Ejemplo

**P1** = $< 2.3, 4.5, -1.2, 0.8 >$
**P2** = $< 1.4, -0.2, 6.7, 4.8 >$

Si usamos **u** = 0.42, $\eta_c = 2$, tenemos:
**H1** = $< 1.42, -0.067, -0.97, 0.9129 >$
**H2** = $< 2.27, 4.37, 6.48, 4.69 >$

# Cruzas para Representación Real

## Recombinación Discreta

Es análoga a la cruza uniforme que se usa con codificación binaria.

Por cada posición *i* del primer hijo, elegimos (con una probabilidad fija *p*) al padre cuyo i-ésimo gene será transmitido a este descendiente.

La posición respectiva del segundo hijo se completa con el valor del gene correspondiente del otro padre.

# Cruzas para Representación Real

## Recombinación Discreta

Si se usa $p$=0.5, los 2 padres contribuirán con la misma cantidad de genes.

## Ejemplo

Si:

**P1** = $(x_1, x_2, x_3, x_4)$

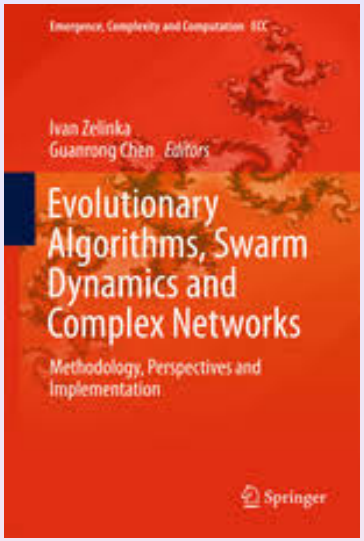
**P2** = $(y_1, y_2, y_3, y_4)$

Usando $p$=0.5, podríamos tener:

**H1** = $(x_1, y_2, y_3, x_4)$

**H2** = $(y_1, x_2, x_3, y_4)$

# Cruzas para Representación Real

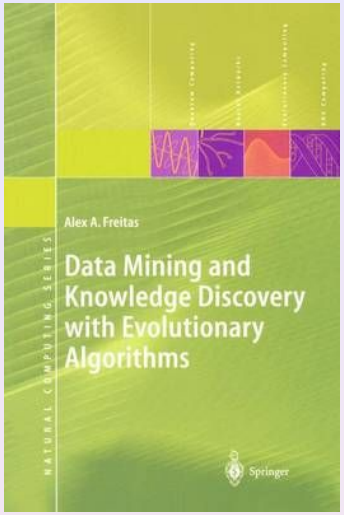



## Recombinación Continua

Al usar este operador (llamado también cruza promedio), algunas posiciones se eligen al azar. Cada posición puede ser seleccionada con una probabilidad fija.

Los genes correspondientes en los descendientes representan la media aritmética de los genes correspondientes de sus padres.

# Cruzas para Representación Real

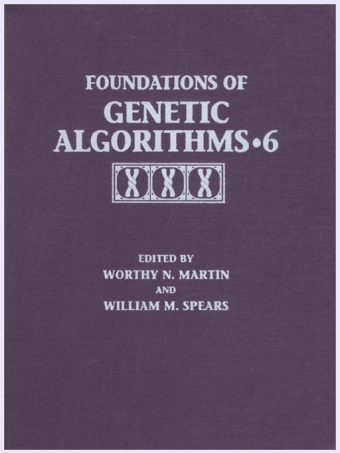



## Recombinación Continua (Ejemplo)

**P1** = $(x_1, x_2, x_3, x_4)$
**P2** = $(y_1, y_2, y_3, y_4)$

Elegimos las posiciones **1** y **4**:
**H1** = $((x_1 + y_1)/2, x_2, x_3, (x_4 + y_4)/2)$
**H2** = $((x_1 + y_1)/2, y_2, y_3, (x_4 + y_4)/2)$

# Cruzas para Representación Real




## Recombinación Continua Completa

Esta cruza produce un solo hijo y es una generalización de la recombinación continua.

Los genes del hijo obtenido contienen el promedio de los genes de sus padres. El i-ésimo gene del descendiente tiene la forma:

$$z_i = \frac{1}{2}(x_i + y_i), \quad i = 1, \ldots, L$$

# Cruzas para Representación Real

## Recombinación con padres múltiples

Eiben et al. [1994] propusieron la **diagonal multi-parent crossover**. Este operador crea $p$ hijos a partir de $p$ padres.

1. Se elige un número de puntos de cruza $N >= 1$.
2. El primer hijo contiene el i-ésimo segmento del padre $i$, para $i = 1, \ldots, p$.
3. Los otros hijos se construyen de una rotación de segmentos de los padres.

A.E. Eiben, P.E. Raué and Zs. Ruttkay, "**Genetic Algorithms with Multi-Parent Recombination**", in Y. Davidor, H.-P. Schwefel and R. Männer (Editors), *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, pp. 78–87, Springer-Verlag, Lecture Notes in Computer Science Vol. 866, 1994.

# Cruzas para Representación Real

## Recombinación basada en aptitud

Algunos investigadores han propuesto el uso de los valores de aptitud para guiar el operador de recombinación, bajo el argumento de que esto acelera la convergencia.

Este tipo de mecanismos se basan en la explotación local del espacio de búsqueda, a diferencia de los tradicionales, que se basan en el azar.

# Cruzas para Representación Real

## Recombinación basada en aptitud

Eiben et al. [1994] consideraron la recombinación multi-padres basada en aptitud.

En esta propuesta, se genera un solo hijo a partir de $p >= 2$ padres.

Cada componente (gene) del hijo se selecciona de uno de los padres con una probabilidad que corresponde a la aptitud relativa del padre.

A.E. Eiben, P.E. Raué and Zs. Ruttkay, "**Genetic Algorithms with Multi-Parent Recombination**", in Y. Davidor, H.-P. Schwefel and R. Männer (Editors), *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, pp. 78–87, Springer-Verlag, Lecture Notes in Computer Science Vol. 866, 1994.

# Cruzas para Representación Real

## Recombinación basada en aptitud

La probabilidad de seleccionar cada componente del padre $x^i$ es:

$$p_i = \frac{f(x^i)}{\sum_{j=1}^{k} f(x^j)}$$

Donde: $f$ es la función de aptitud y $K$ es el tamaño de la población de padres.

# Cruzas para Representación Real

## Cruza Heurística

Este operador fue propuesto por Wright [1991] y produce un único hijo $z$ de los padres $x$, $y$.

Supongamos que $y$ no es peor que $x$. O sea (suponiendo maximización), $f(y) >= f(x)$.

En este caso, el hijo es: $z = y + u(y - x)$

Donde $u$ es una variable aleatoria en el rango $[0, 1]$

Alden H. Wright, "**Genetic Algorithms for Real Parameter Optimization**", in Gregory J.E. Rawlins (Editor), *Foundations of Genetic Algorithms*, pp. 205–218, Morgan Kaufmann Publishers, San Mateo, California, USA, 1991.

# Cruzas para Representación Real

## Cruza Heurística

La cruza heurística usa información local para determinar una dirección de búsqueda.

Se hace notar que este operador de cruza puede verse como una variante aleatoria del procedimiento de optimización denominado **descenso empinado** (*steepest descent*).

# Mutación

Se considera como un operador secundario en los algoritmos genéticos canónicos.

Se suelen recomendar porcentajes de mutación entre 0.001 y 0.01 para la representación binaria.

Algunos investigadores han sugerido que el usar porcentajes altos de mutación al inicio de la búsqueda, y luego decrementarlos exponencialmente, favorece el desempeño de un algoritmo genético [Fogarty, 1989].

Terence C. Fogarty, "**Varying the Probability of Mutation in the Genetic Algorithm**", in J. David Schaffer (Editor), *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 104–109, Morgan Kaufmann Publishers, San Mateo, California, USA, 1989.

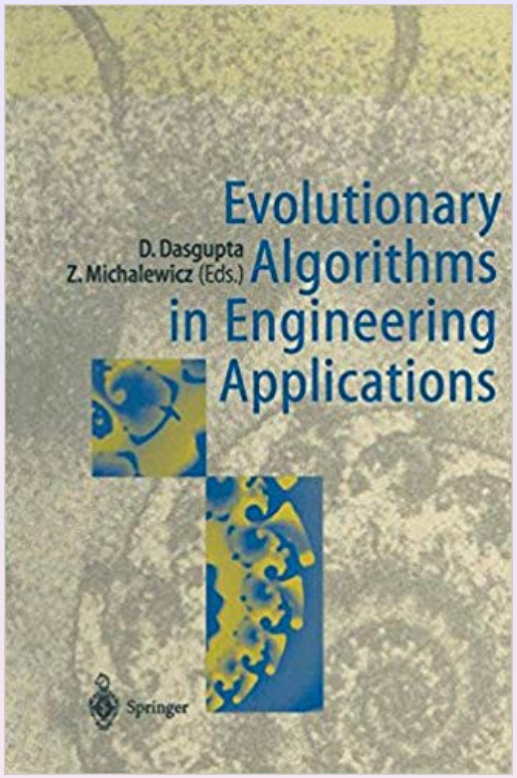
# Mutación

Algunos autores sugieren que $p_m = \frac{1}{L}$ (donde $L$ es la longitud de la cadena cromosómica) es un límite inferior para el porcentaje óptimo de mutación [Bäck, 1993].

El papel que juega la mutación en el proceso evolutivo, así como su comparación con la cruza, sigue siendo tema frecuente de investigación y debate en la comunidad de computación evolutiva.

Thomas Bäck, "**Optimal Mutation Rates in Genetic Search**" in Stephanie Forrest (Editor), *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 2–8, Morgan Kaufmann Publishers, San Mateo, California, USA, July 1993.

# Operadores de Mutación




- Permutaciones
- Programación Genética
- Representación Real

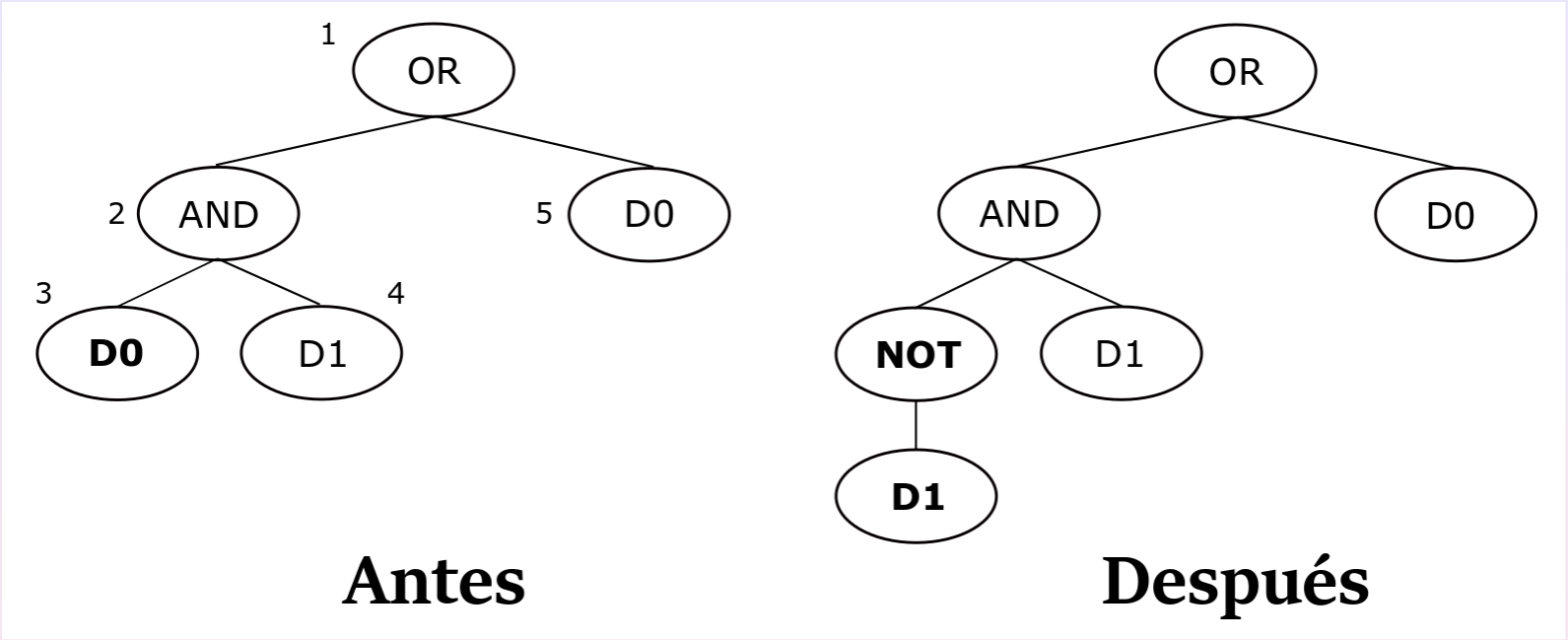
# Mutación para Programación Genética

La **mutación** es un operador asexual que, en el caso de la programación genética, se aplica sobre una sola expresión $S$.

Se selecciona aleatoriamente un punto de mutación en el árbol y se remueve todo lo que esté en dicho punto y abajo de él, insertándose en su lugar un sub-árbol generado aleatoriamente.

La profundidad máxima de este sub-árbol está acotada por la profundidad total permitida para cualquier expresión $S$ en la población inicial.

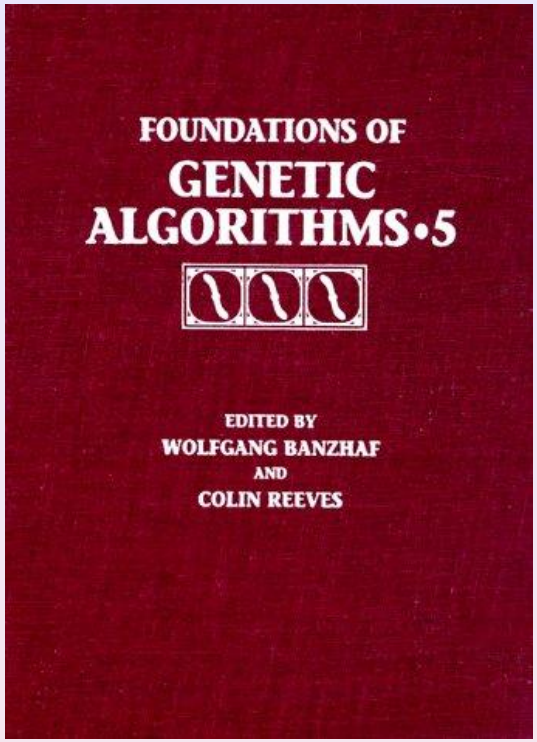
# Mutación para Programación Genética




## Ejemplo

En este caso, suponemos que el punto de mutación es el nodo 3.

# Mutación para Representación Real




- No Uniforme
- De Límite
- Uniforme
- Parameter-Based Mutation

# Mutación para Representación Real

## Mutación No Uniforme

Propuesta por Michalewicz [1992].

Dado: $P =< V_1, \ldots, V_m >$ el individuo mutado será: $P' =< V_1, \ldots, V'_k, \ldots, V_m >$

donde:

$$V'_k = \begin{cases} V_k + \Delta(t, UB - V_k) & \text{si } R = \text{Cierto} \\ V_k - \Delta(t, V_k - LB) & \text{si } R = \text{Falso} \end{cases} \quad (3)$$

y la variable $V_k$ está en el rango $[LB, UB]$.

Zbigniew Michalewicz, "**Genetic Algorithms + Data Structures = Evolution Programs**", Springer-Verlag, Second Edition, 1992.

# Mutación para Representación Real

## Mutación No Uniforme

$$R = flip(0.5)$$

$\Delta(t, y)$ regresa un valor en el rango $[0, y]$ tal que la probabilidad de que $\Delta(t, y)$ esté cerca de cero se incrementa conforme $t$ (generación actual) crece.

Esto hace que este operador explore de manera más global el espacio de búsqueda al inicio (cuando $t$ es pequeña) y de manera más local en etapas posteriores.

# Mutación para Representación Real

## Mutación No Uniforme

Michalewicz sugiere usar:

$$\Delta(t, y) = y \cdot (1 - r^{(1-\frac{t}{T})^b})$$

Donde:
$r$ es un número aleatorio real entre 0 y 1, $T$ es el número máximo de generaciones y $b$ es un parámetro que define el grado de no uniformidad de la mutación (Michalewicz sugiere usar $b = 5$).

## Mutación No Uniforme (Ejemplo)

Supongamos que tenemos al individuo siguiente a ser mutado:
$\mathbf{P} = < 2.3, 4.5, -1.2, 0.8 >$

$V_k$ = 4.5, $l_k$ = -2.0, $u_k$ = 6.5, $T$ = 50, $t$ = 5,
$R$ = Falso, $r$ = 0.24, $b$ = 5.

$V'_k = V_k - \Delta(t, V_k - l_K)$ = 4.5 - $\Delta(5, 4.5 + 2)$ = 4.5 - $\Delta$ (5, 6.5)

$\Delta(5, 6.5) = 6.5(1 - 0.24)^{(1-\frac{5}{50})^5}) = 6.489435$
$V'_k$ = 4.5 - 6.489435 = -1.989435

# Mutación para Representación Real

## Mutación de L'ımite

Dado: $\mathbf{P} = < V_1, \ldots, V_m >$
el individuo mutado será: $P' =< V_1, \ldots, V'_k, \ldots, V_m >$

Donde:

$$V'_k = \begin{cases} LB & \text{si } flip(0.5) = \text{TRUE} \\ UB & \text{de lo contrario} \end{cases} \quad (4)$$

y $[LB, UB]$ definen los rangos mínimos y máximos permisibles de valores para la variable $V'_k$.

# Mutación para Representación Real

## Mutación de L'ımite (Ejemplo)

Supongamos que el individuo a ser mutado es:
$\mathbf{P} = < 1.5, 2.6, -0.5, 3.8 >$

$V'_k =$ -0.5, $LB =$ -3.0, $UB =$1.3

Supongamos que: flip(0.5) = TRUE

$V'_k =$ -3.0

# Mutación para Representación Real

## Mutación Uniforme

Dado:
**P** = $< V_1, \ldots, V_m >$

el individuo mutado será: **P'** = $< V_1, \ldots, V'_k, \ldots, V_m >$

donde:

$$V'_k = rnd(LB, UB)$$

Se usa una distribución uniforme y $[LB, UB]$ define los rangos mínimos y máximos de la variable $V'_k$.

# Mutación para Representación Real

## Mutación Uniforme (Ejemplo)

Supongamos que tenemos el siguiente individuo a ser mutado:
$\mathbf{P} = < 5.3, -1.3, 7.8, 9.1 >$

$V_k = 5.3, \quad \text{LB} = 0.0, \quad \text{UB} = 10.5$

$V'_k = \text{rnd}(0.0, 10.5) \qquad V'_k = 4.3$

# Mutación para Representación Real

## Parameter-Based Mutation

Este operador de mutación, propuesto por Deb [1995] se suele usar en combinación con la *Simulated-Based Crossover* (SBX).

Su objetivo es crear una solución $c$ en la vecindad de una solución padre $y$. Se presupone que el padre $y$ está acotado ($y \in [y_l, y_u]$).

## Algoritmo (Parte 1)

1. Crear un número aleatorio $u$ entre 0 y 1.
2. Calcular:

$$\delta_q = \begin{cases} \left[2u + (1-2u)(1-\delta)^{\eta_m+1}\right]^{\frac{1}{\eta_m+1}} - 1 & \text{si } u \leq 0.5 \\ 1 - \left[2(1-u) + 2(u-0.5)(1-\delta)^{\eta_m+1}\right]^{\frac{1}{\eta_m+1}} & \text{de lo contrario} \end{cases} \quad (5)$$

Donde $\delta = \min\left[(y - y_l), (y_u - y)\right]/(y_u - y_l)$.

# Mutación para Representación Real

## Parameter-Based Mutation

El parámetro $\eta_m$ es el índice de distribución para la mutación y toma cualquier valor no negativo.

Deb sugiere usar:

$$\eta_m = 100 + t \quad (t = \text{generación actual})$$

## Algoritmo (Parte 2)

3. El valor de la posición mutada se determina usando:

$$y' = y + \delta_q \Delta_{max}$$

donde $\Delta_{max}$ es la máxima perturbación permitida:

$$\Delta_{max} = y_u - y_l$$

considerando que:

$$y \in [y_l, y_u]$$

# Mutación para Representación Real

## Parameter-Based Mutation (Ejemplo Parte 1)

Supongamos que el individuo a mutarse es:
**P** = $< 2.3, 4.5, -1.2, 0.8 >$

$y$ = -1.2, $u = 0.72$, $t$ =20
$y_l$ = -2.0, $y_u = 6.0$
$\eta_m = 100 + t = 120$

$\delta = \min\ [(-1.2 + 2.0), (6.0 + 1.2)]\ /(6.0 + 2.0)$
$\delta = 0.8/8.0 = 0.1$

# Mutación para Representación Real

## Parameter-Based Mutation (Ejemplo Parte 2)

$$\delta_q = 1 - \left[2(1 - 0.72) + 2(0.72 - 0.5)(1 - 0.1)^{121}\right]^{\frac{1}{121}}$$

$$\delta_q = 4.7804 \times 10^3$$

$$\Delta_{max} = y_u - y_l = 6.0 + 2.0 = 8.0$$

$$y' = \text{-1.2 + 0.03824 = -1.16175}$$

# Cruza vs. Mutación

La cruza uniforme es más "explorativa" que la cruza de un punto.

Por ejemplo, dados:

$$P1 = 1 * * * * 1$$

$$P2 = 0 * * * * 0$$

La cruza uniforme producirá individuos del esquema $* * * * * **$, mientras que la cruza de un punto producirá individuos de los esquemas $1 * * * * 0$ y $0 * * * * 1$.

# Cruza vs. Mutación

¿Cuál es el poder exploratorio de la mutación?

Si el porcentaje de mutación es cero, no hay alteración alguna.

Si el porcentaje de mutación es uno, la mutación crea siempre complementos del individuo original.

Si el porcentaje de mutación es 0.5, hay una alta probabilidad de alterar fuertemente el esquema de un individuo.

En otras palabras, podemos controlar el poder de alteración de la mutación y su capacidad de exploración puede hacerse equivalente a la de la cruza.

# Cruza vs. Mutación

El tipo de exploración efectuada por la mutación es, sin embargo, diferente a la de la cruza.

Por ejemplo, dados:

$$P1 = 10 * * * *$$

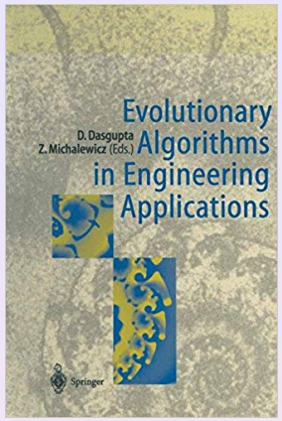
$$P2 = 11 * * * *$$

La cruza producirá sólo instancias del esquema $1 * * * **$.

El primer "1" en el esquema está garantizado (sin importar qué tipo de cruza se use), porque es común en los esquemas de ambos padres.

La mutación, sin embargo, no respetará necesariamente este valor.

# Cruza vs. Mutación

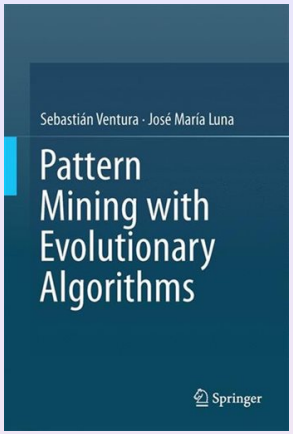



La cruza “preserva” los alelos que son comunes en los 2 padres.

Esta preservación limita el tipo de exploración que la cruza puede realizar.

Esta limitación se agudiza conforme la población pierde diversidad, puesto que el número de alelos comunes se incrementará.

# Cruza vs. Mutación

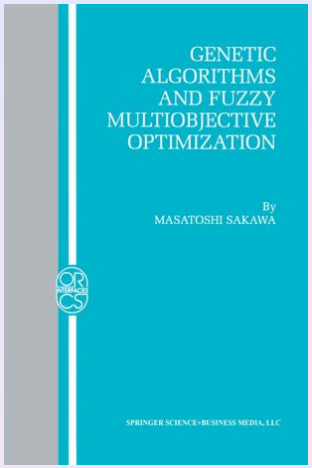



Cuando buscamos localizar el óptimo global de un problema, la mutación puede ser más útil que la cruza.

Si lo que nos interesa es ganancia acumulada (el objetivo original del algoritmo genético), la cruza es entonces preferible.

La cruza parece trabajar bien con funciones que están altamente correlacionadas o tienen epístasis moderada.

# Ajuste de Parámetros de un Algoritmo Genético




Los parámetros principales de un algoritmo genético son:

- Tamaño de población
- Porcentaje de cruza
- Porcentaje de mutación

Estos parámetros normalmente interactúan entre sí de forma no lineal, por lo que no pueden optimizarse de manera independiente.

# Ajuste de Parámetros de un Algoritmo Genético

De Jong [1975] efectuó una serie de experimentos para comparar algoritmos genéticos con técnicas de gradiente.

En su estudio, De Jong propuso cinco funciones de prueba que exhibían una serie de características que las hacían difíciles para las técnicas de gradiente.

A. K. De Jong, "**An Analysis of the Behavior of a Class of Genetic Adaptive Systems**", PhD thesis, University of Michigan, Ann Arbor, USA, 1975.

# Ajuste de Parámetros de un Algoritmo Genético

Sin embargo, antes de proceder a realizar sus experimentos, De Jong decidión analizar la influencia de los parámetros de un algoritmo genético en su desempeño.

# Ajuste de Parámetros de un Algoritmo Genético

Para medir el impacto de los parámetros de un AG, De Jong propuso dos métricas:

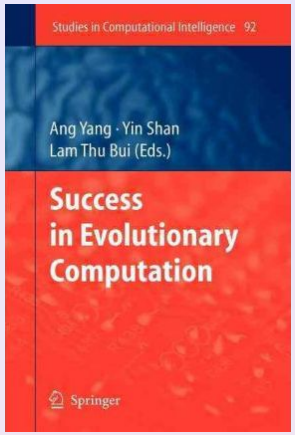
1. **Desempeño en Línea** (Online): Es la aptitud promedio de todos los individuos que han sido evaluados en las últimas $t$ generaciones.
2. **Desempeño fuera de Línea** (Offline): Es el promedio de las mejores aptitudes evaluadas en las últimas $t$ generaciones.

# Ajuste de Parámetros de un Algoritmo Genético

Para que un algoritmo de búsqueda tenga un buen desempeño **en línea**, debe decidir rápidamente dónde están las regiones más prometedoras de búsqueda y concentrar ahí sus esfuerzos.

El desempeño **fuera de línea** no penaliza al algoritmo de búsqueda por explorar regiones pobres del espacio de búsqueda, siempre y cuando ello contribuya a alcanzar las mejores soluciones posibles (en términos de aptitud).

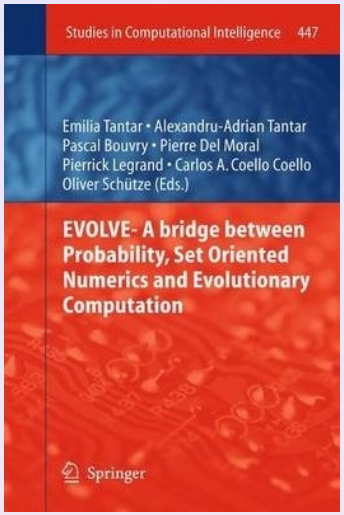
# Ajuste de Parámetros de un Algoritmo Genético




Los parámetros hallados por De Jong que proporcionaron el mejor desempeño tanto en línea como fuera de línea fueron los siguientes:

| | |
|---|---|
| Tamaño de la población | 50 a 100 individuos |
| Porcentaje de cruza | 0.60 |
| Porcentaje de mutación | 0.001 |

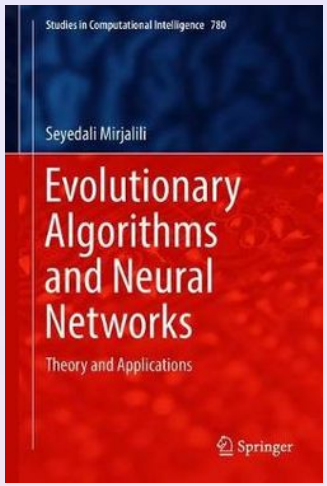
# Ajuste de Parámetros de un Algoritmo Genético




Algunas conclusiones interesantes de De Jong [1975] fueron las siguientes:

1. Incrementar el tamaño de la población reduce los efectos estocásticos del muestreo aleatorio en una población finita, por lo que mejora el desempeño del algoritmo a largo plazo, aunque esto es a costa de una respuesta inicial más lenta.

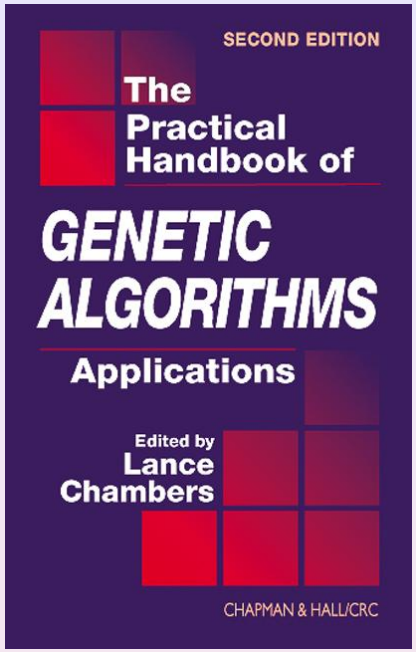
# Ajuste de Parámetros de un Algoritmo Genético




2. Incrementar el porcentaje de mutación mejora el desempeño **fuera de línea** a costa de sacrificar el desempeño **en línea**.

3. Reducir el porcentaje de cruza mejora la media de desempeño, lo que sugiere que producir una generación de individuos completamente nuevos no es bueno.

# Ajuste de Parámetros de un Algoritmo Genético




4. Observando el desempeño de diferentes operadores de cruza, De Jong concluyó que, aunque el incrementar el número de puntos de cruza afecta su disrupción de esquemas desde una perspectiva teórica, esto no parece tener un impacto significativo en la práctica.

# Ajuste de Parámetros de un Algoritmo Genético

Grefenstette [1986] usó un algoritmo genético para optimizar los parámetros de otro (un meta-algoritmo genético).

El meta-algoritmo genético fue usado para evolucionar unos 50 conjuntos de parámetros de un algoritmo genético que se usó para resolver las funciones de De Jong.

John J. Grefenstette, "**Optimization of Control Parameters for Genetic Algorithms**", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-16, No. 1, pp. 122–128, January/February 1986.

# Ajuste de Parámetros de un Algoritmo Genético

En el estudio de Grefenstette, cada individuo codificaba seis parámetros:

1. Tamaño de la población
2. Porcentaje de cruza
3. Porcentaje de mutación
4. Intervalo generacional (porcentaje de la población que se reemplaza en cada generación)
5. Ventana de escalamiento (sin escalamiento, escalamiento basado en $f(x)$ mínima de la primera generación, escalamiento basado en la $f(x)$ mínima de las últimas $W$ generaciones.
6. Estrategia de selección (elitista o puramente seleccionista).

La aptitud de un individuo era una función de su desempeño en línea y fuera de línea.

# Ajuste de Parámetros de un Algoritmo Genético

El meta-algoritmo genético usaba los parámetros de De Jong, y con él, Grefenstette [1986] obtuvo los siguientes valores óptimos de los parámetros para el desempeño **en línea**:

| | |
|---|---|
| Tamaño de la población: | 30 individuos |
| Porcentaje de cruza: | 0.95 |
| Porcentaje de mutación: | 0.01 |
| Selección: | Elitista |
| Intervalo generacional: | 1.0 (100%) |
| Ventana de escalamiento: | 1 (basado en la $f(x)$ mínima de la primera generación) |

# Ajuste de Parámetros de un Algoritmo Genético

Estos parámetros mejoraron ligera, pero no significativamente el desempeño **en línea** del algoritmo genético con respecto a los de De Jong.

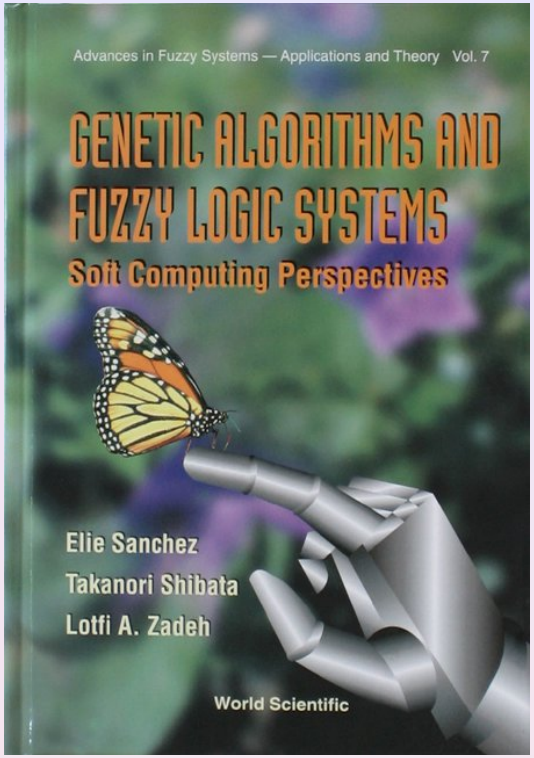
Sin embargo, Grefenstette no pudo mejorar el desempeño **fuera de línea**.

# Ajuste de Parámetros de un Algoritmo Genético

Algunas observaciones realizadas por Grefenstette [1986] fueron las siguientes:

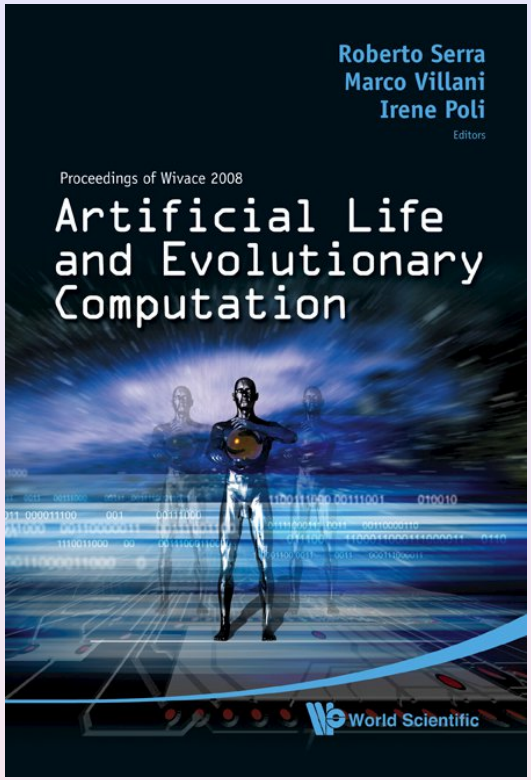
1. Los porcentajes de mutación por encima de 0.05 tienden a ser perjudiciales con respecto al desempeño **en línea**, y el algoritmo genético aproxima el comportamiento de la búsqueda aleatoria para porcentajes de mutación $\geq .1$, sin importar qué otros parámetros se usen.

# Ajuste de Parámetros de un Algoritmo Genético



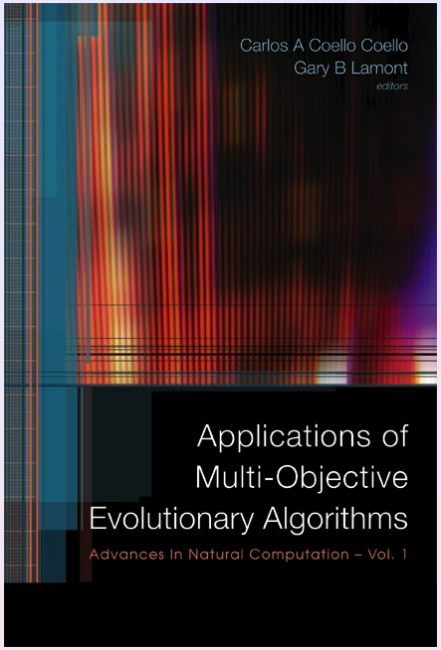

2. La ausencia de mutación está también asociada con un desempeño pobre del algoritmo genético, lo que sugiere que su papel es más importante de lo que normalmente se creía en aquel entonces, pues permite refrescar valores perdidos del espacio de búsqueda.

# Ajuste de Parámetros de un Algoritmo Genético



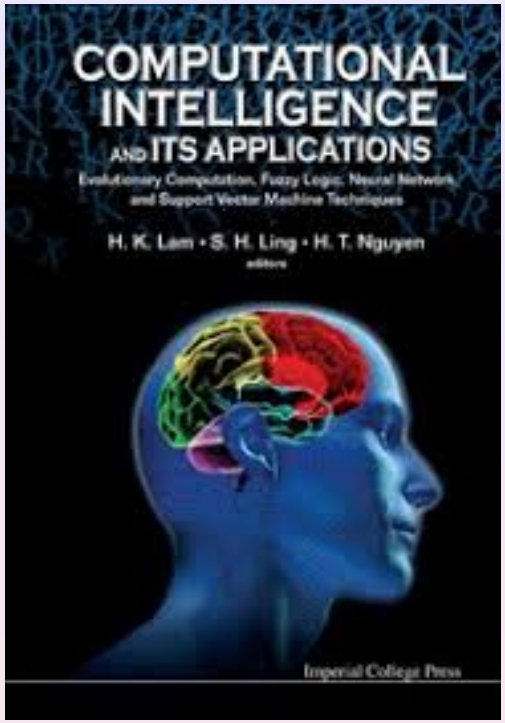

3. El tamaño óptimo de la población para lograr el mejor desempeño posible **fuera de línea** está entre 60 y 110 individuos. Un alto intervalo generacional y el uso de una estrategia elitista también mejoran el desempeño del algoritmo genético.

# Ajuste de Parámetros de un Algoritmo Genético




4. Para poblaciones pequeñas (20 a 40 individuos), el buen desempeño **en línea** está asociado con un porcentaje alto de cruza combinado con un porcentaje bajo de mutación o viceversa (un porcentaje bajo de cruza combinado con un porcentaje alto de mutación).

# Ajuste de Parámetros de un Algoritmo Genético




5. Para poblaciones de tamaño medio (30 a 90 individuos), el porcentaje óptimo de cruza parece decrementarse conforme se aumenta el tamaño de la población.

# Ajuste de Parámetros de un Algoritmo Genético

Goldberg [1985] realizó un estudio teórico del tamaño ideal de la población de un algoritmo genético en función del número esperado de nuevos esquemas por miembro de la población.

Usando una población inicial generada aleatoriamente con igual probabilidad para el cero y el uno, Goldberg derivó la siguiente expresión:

$$\text{Tam\_Poblacion} = 1.65^{2^{0.21L}} \tag{6}$$

donde: $L$ = longitud de la cadena (binaria).

David E. Goldberg, "**Optimal initial population size for binary-coded genetic algorithms**", Technical Report No. 85001, Tuscaloosa, University of Alabama, The Clearlinghouse for Genetic Algorithms, USA, 1985.

# Ajuste de Parámetros de un Algoritmo Genético

Esta expresión sugiere tamaños de población demasiado grandes para cadenas de longitud moderada.

## Ejemplos

$L = 30$, Tam_Población = **130**

$L = 40$, Tam_Población = **557**

$L = 50$, Tam_Población = **2389**

$L = 60$, Tam_Población = **10244**

# Ajuste de Parámetros de un Algoritmo Genético


Hubieron innumerables ataques al trabajo de Goldberg antes mencionado, ya que éste se basó en una interpretación errónea del teorema de los esquemas.

Para entender la falacia del argumento de Goldberg, debemos comenzar por definir un concepto muy importante de computación evolutiva: el **paralelismo implícito**.

# Ajuste de Parámetros de un Algoritmo Genético

## Definición de Paralelismo Implícito

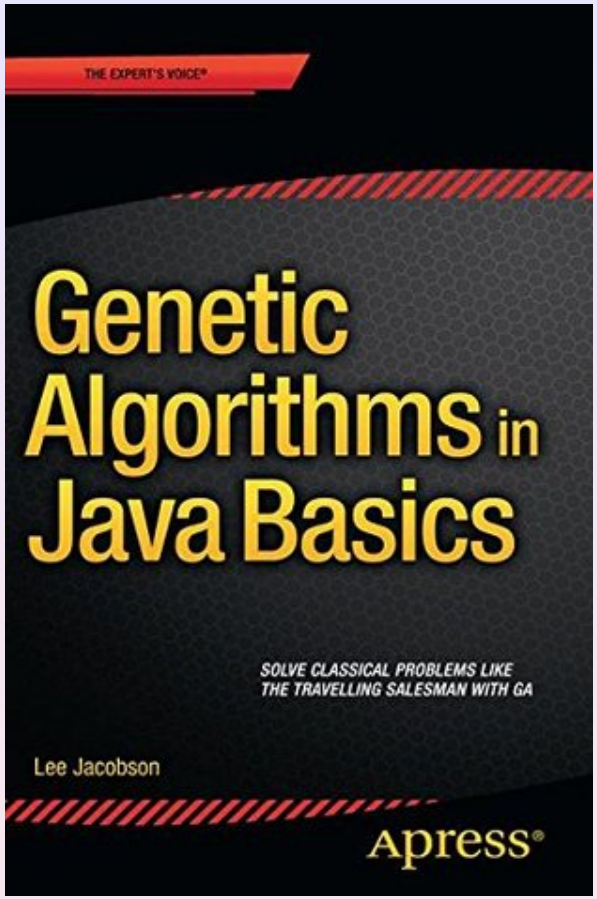
Mientras un algoritmo genético calcula explícitamente las aptitudes de los $N$ miembros de una población, al mismo tiempo estima implícitamente las aptitudes promedio de una cantidad mucho mayor de esquemas, calculando implícitamente las aptitudes promedio observadas de los esquemas que tienen instancias en la población.

# Ajuste de Parámetros de un Algoritmo Genético

Según el teorema de los esquemas (que veremos más adelante), un algoritmo genético procesa $O(N^3)$ esquemas.

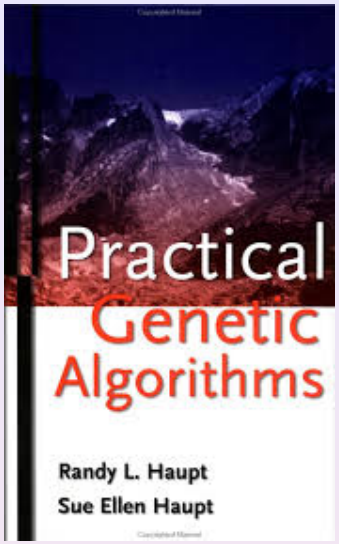
A partir de esta idea, Goldberg concluye entonces que a mayor valor de $N$ (tamaño de la población), mejor desempeño tendrá el algoritmo genético, y de ahí deriva su expresión para calcular el tamaño óptimo de una población.

# Ajuste de Parámetros de un Algoritmo Genético




El problema de este argumento es que sólo hay $3^L$ esquemas en una representación binaria, por lo que no se pueden procesar $O(N^3)$ esquemas si $N^3$ es mucho mayor que $3^L$.

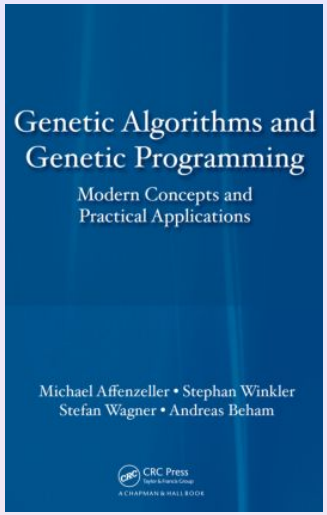
# Ajuste de Parámetros de un Algoritmo Genético




Robertson [1988] determinó que en los algoritmos genéticos paralelos, el desempeño se incrementaba monotónicamente con el tamaño de la población sin seguir la expresión exponencial de Goldberg.

George G. Robertson, "**Population size in classifier systems**", in *Proceedings of the Fifth International Conference on Machine Learning*, pp. 142–152, Morgan Kaufmann Publishers, 1988.

# Ajuste de Parámetros de un Algoritmo Genético




Otros investigadores han derivado expresiones según las cuales, un incremento lineal del tamaño de la población corresponde con un buen desempeño del algoritmo genético.

La regla empírica más común es usar una población de al menos 2 veces $L$.

# Ajuste de Parámetros de un Algoritmo Genético

Algunas observaciones de Goldberg [1989] son las siguientes:

1. Cuando puede demostrarse velocidad convergencia de un algoritmo genético, ésta parece no ser peor que una función cuadrática o cúbica del número de bloques constructores del problema, independientemente del tipo de esquema de solución utilizado.
2. La teoría sugiere que el tamaño óptimo de la población es $N = 3$, sin importar la longitud de la cadena cromosómica. Esta observación dio pie al micro-algoritmo genético [Krishnakumar, 1989].

David E. Goldberg, "**Sizing Populations for Serial and Parallel Genetic Algorithms**", in J. David Schaffer (Editor), *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 70–79, Morgan Kaufmann Publishers, San Mateo, California, USA, 1989.

K. Krishnakumar, "**Micro-genetic algorithms for stationary and non-stationary function optimization**", *SPIE Proceedings: Intelligent Control and Adaptive Systems*, Vol. 1196, pp. 289–296, 1989.

## Micro-Algoritmos Genéticos

El funcionamiento de un micro-algoritmo genético es el siguiente:

1. Generar al azar una población muy pequeña.
2. Aplicar los operadores genéticos hasta lograr convergencia nominal (por ejemplo, hasta que todas las cadenas sean iguales o muy similares).
3. Generar una nueva población transfiriendo los mejores individuos de la población anterior a la nueva, y generando al azar los individuos restantes.
4. Continuar hasta que ya no haya mejora (se presupone que en este caso, ya se convergió al óptimo).