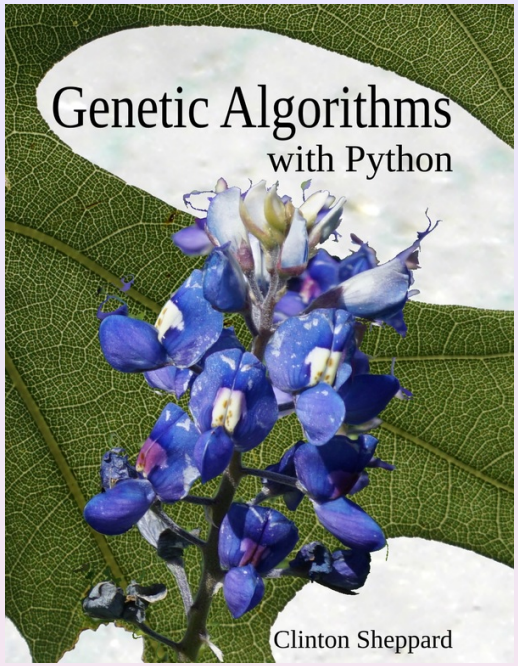
Introducción a la Computación Evolutiva

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 4



Se llama **operador de reproducción** a todo aquel mecanismo que influencia la forma en que se pasa la información genética de padres a hijos. Los operadores de reproducción caen en 3 amplias categorías:

(a) Cruza

(b) Mutación

(c) Reordenamiento

La **cruza** es un operador que forma un nuevo cromosoma combinando partes de cada uno de sus cromosomas padres.

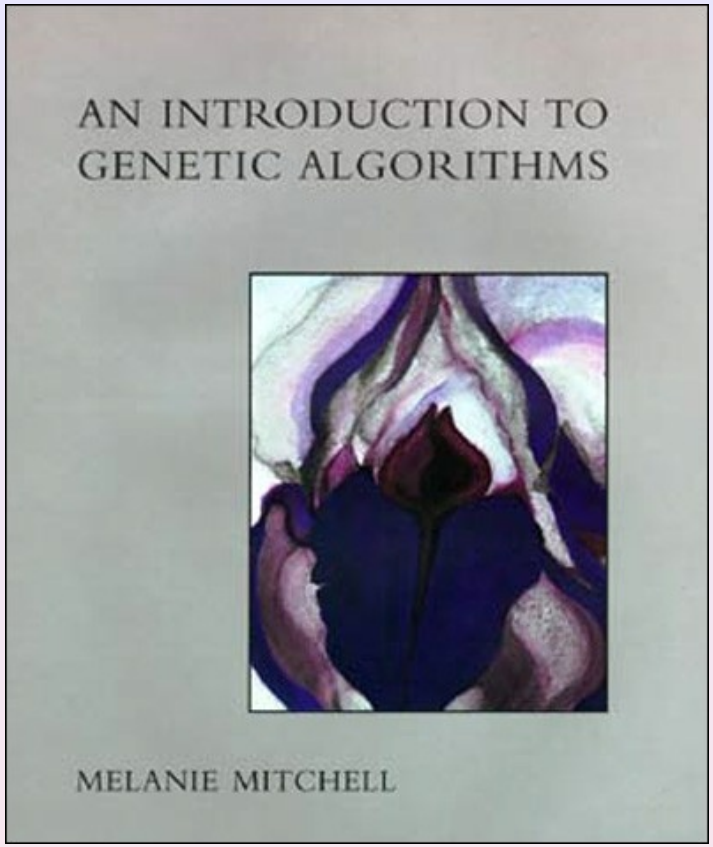
Before Mutation

A5	1	1	1	0	0	0

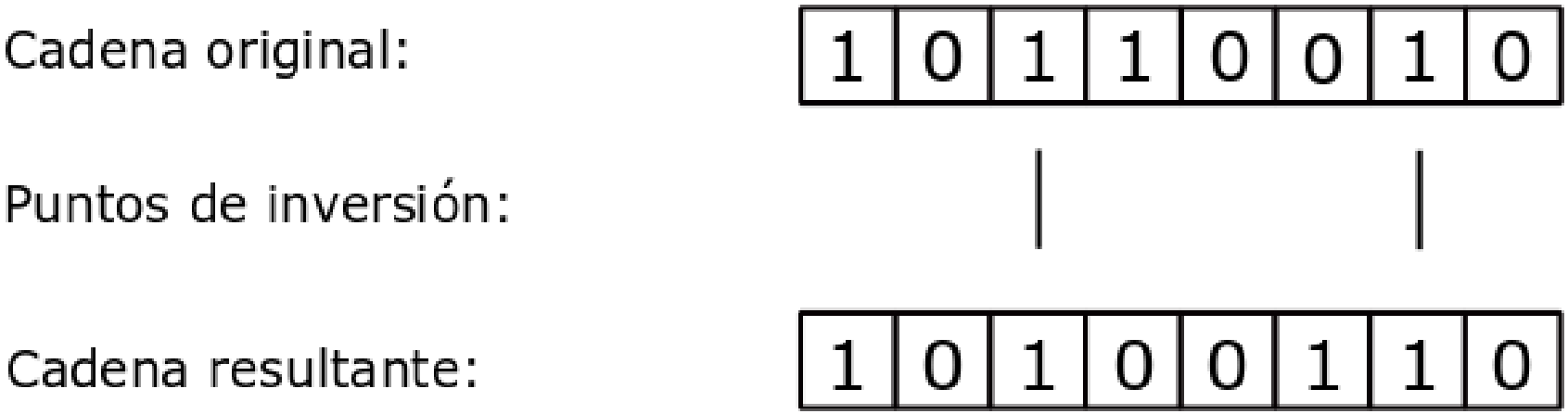
After Mutation

A5	1	1	0	1	1	0

Se denomina **mutación** a un operador que forma un nuevo cromosoma a través de alteraciones (usualmente pequeñas) de los valores de los genes de un solo cromosoma padre.



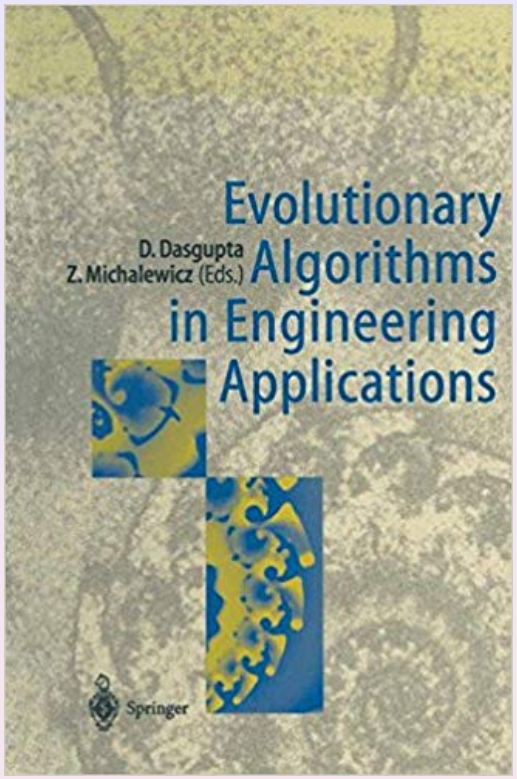
Un **operador de reordenamiento** es aquél que cambia el orden de los genes de un cromosoma, con la esperanza de juntar los genes que se encuentren relacionados, facilitando así la producción de bloques constructores.



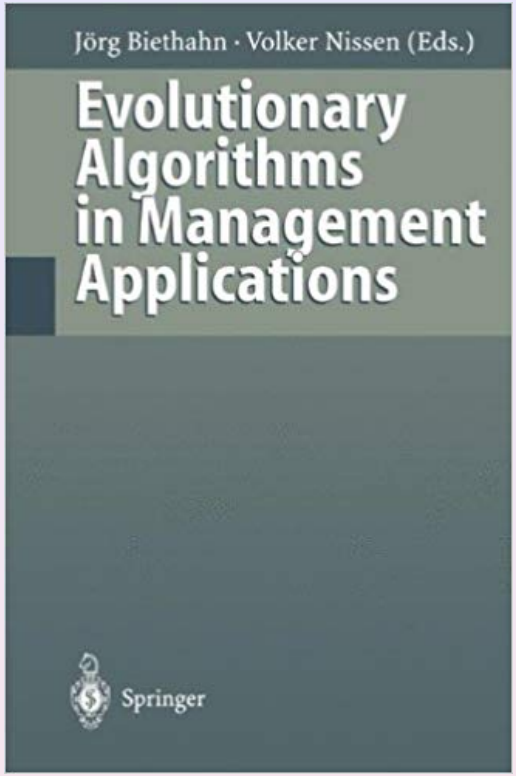
La **inversión** es un ejemplo de un operador de reordenamiento en el que se invierte el orden de todos los genes comprendidos entre 2 puntos seleccionados al azar en el cromosoma.

En un algoritmo genético, cuando una población no tiene **variedad requisito**, la cruza no será útil como operador de búsqueda, porque tendrá propensión a simplemente regenerar a los padres.

Conceptos de Computación Evolutiva

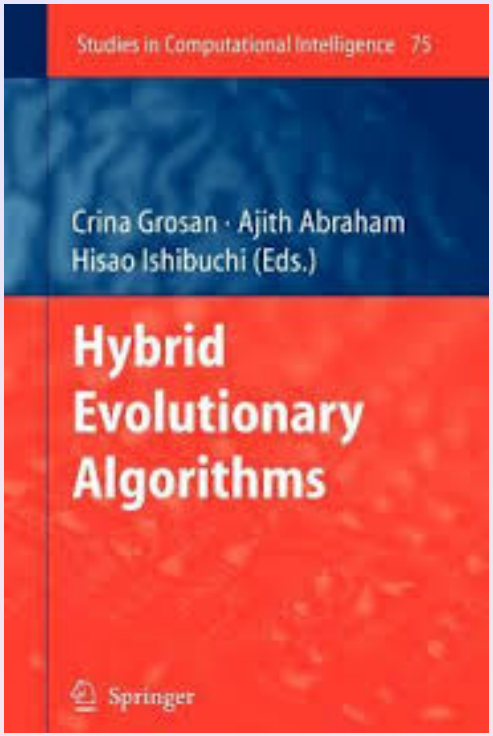


Es importante aclarar que en los algoritmos genéticos, los operadores de reproducción actúan sobre los **genotipos** y no sobre los **fenotipos** de los individuos.



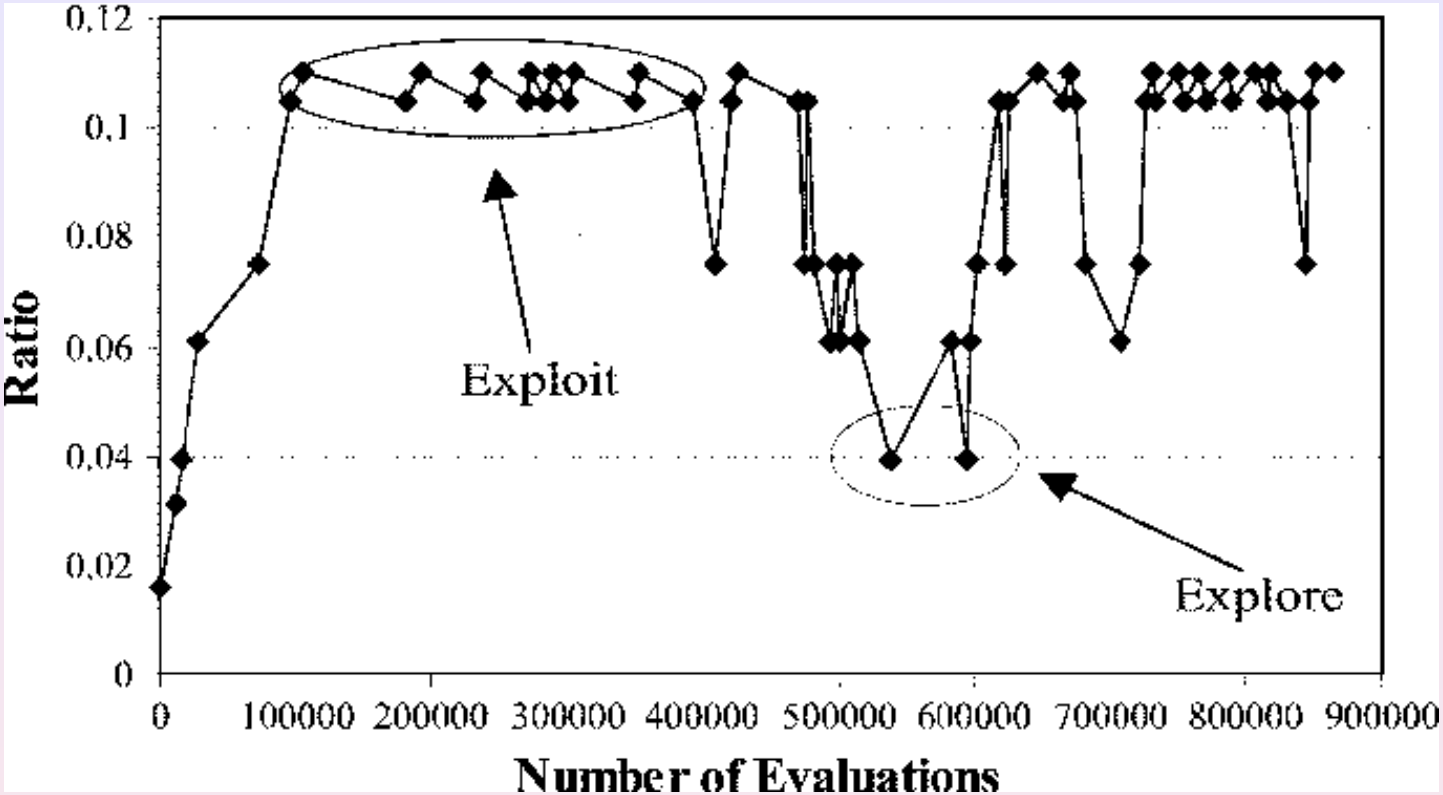
Se denomina **elitismo** al mecanismo utilizado en los algoritmos genéticos para asegurar que los cromosomas de los miembros más aptos de una población se pasen a la siguiente generación sin ser alterados por ningún operador genético.

Conceptos de Computación Evolutiva



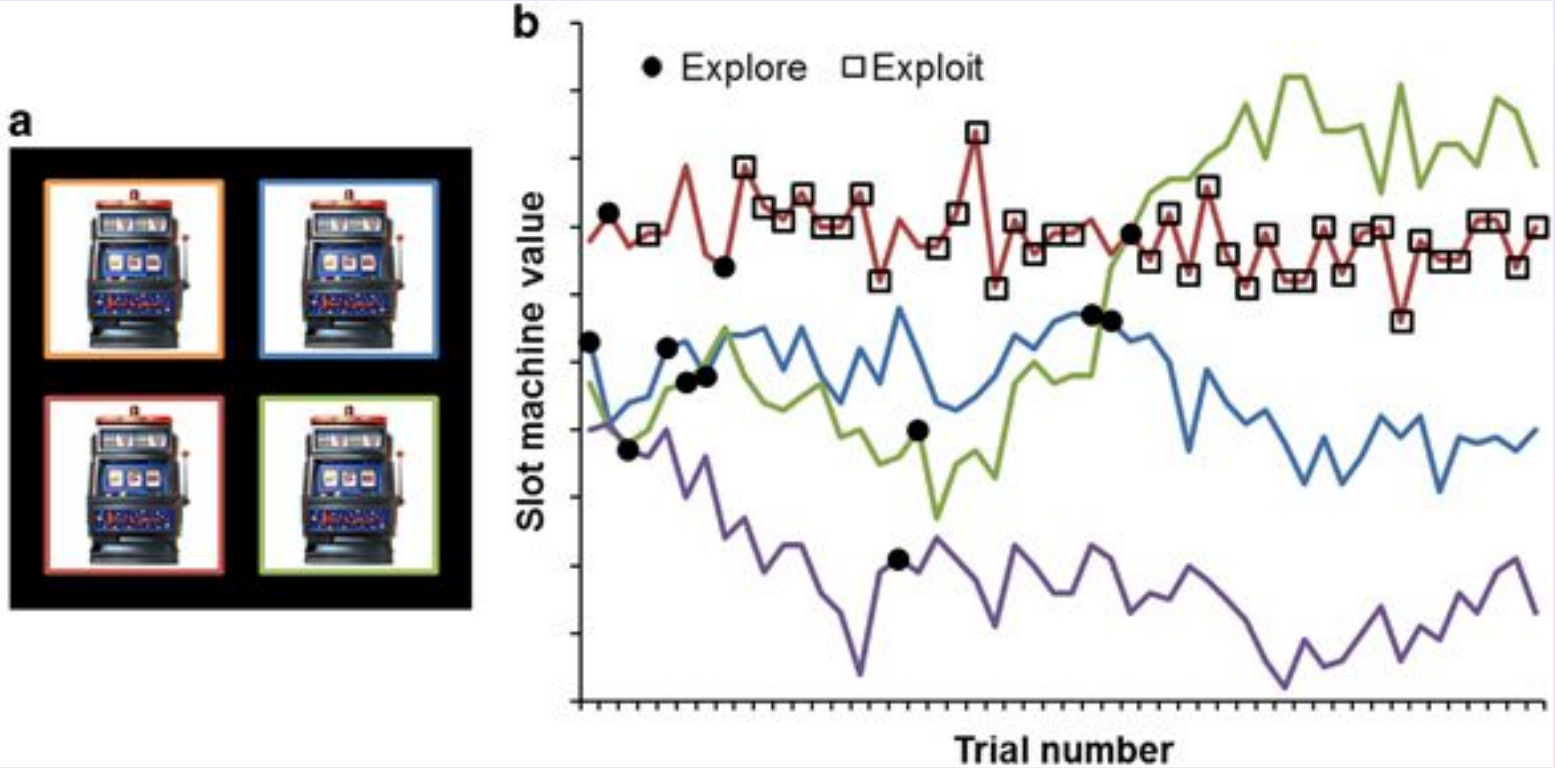
Usar **elitismo** asegura que la aptitud máxima de la población nunca se reducirá de una generación a la siguiente. Sin embargo, no necesariamente mejora la posibilidad de localizar el óptimo global de una función.

Conceptos de Computación Evolutiva



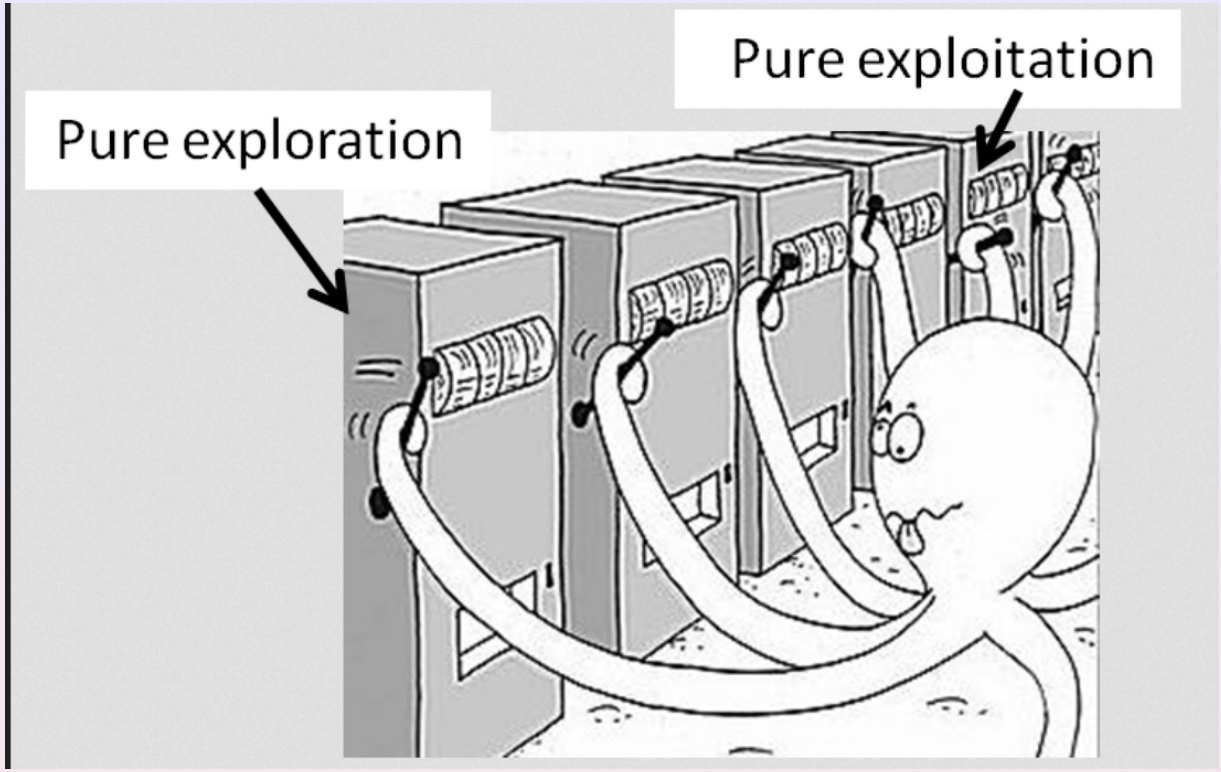
Cuando se atraviesa un espacio de búsqueda, se denomina **explotación** al proceso de usar la información obtenida de los puntos visitados previamente para determinar qué lugares resulta más conveniente visitar a continuación.

Conceptos de Computación Evolutiva



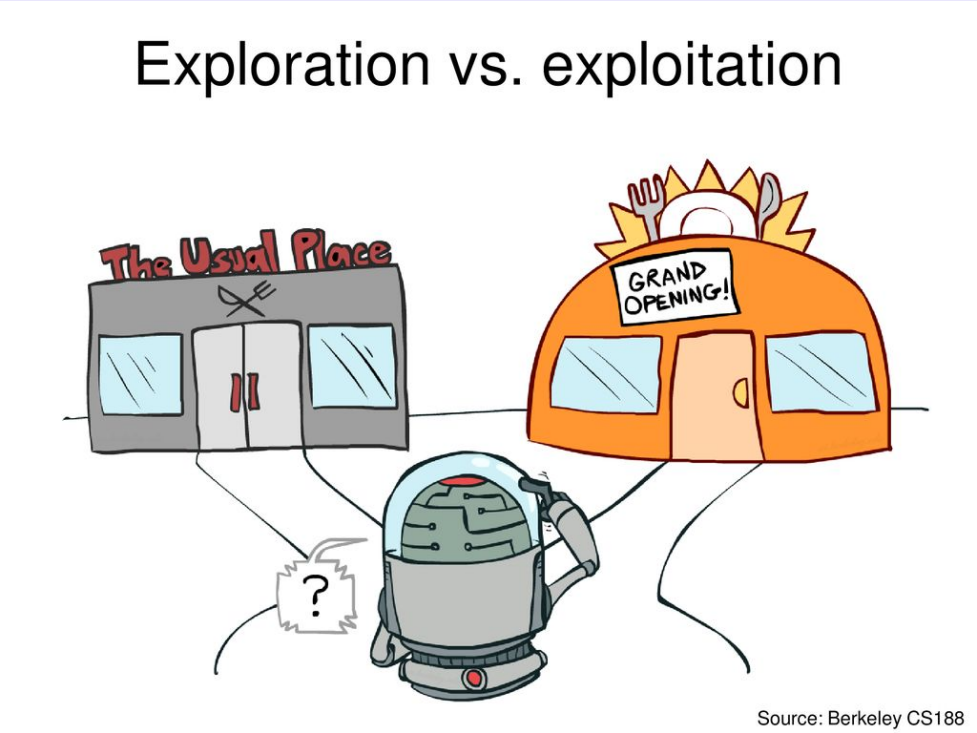
Se denomina **exploración** al proceso de visitar regiones del espacio de búsqueda completamente nuevas, para ver si puede encontrarse algo prometedor.

Conceptos de Computación Evolutiva



La **exploración** involucra grandes saltos hacia lo desconocido.

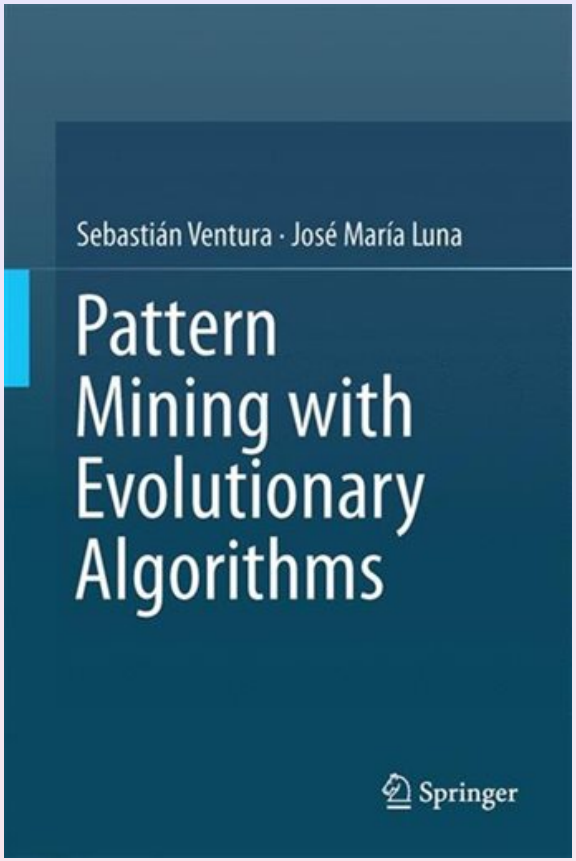
La **explotación** normalmente involucra movimientos finos.



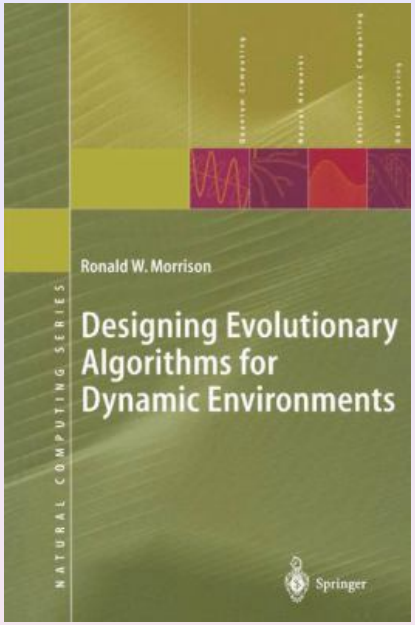
La **explotación** es buena para encontrar óptimos locales.

La **exploración** es buena para evitar quedar atrapado en óptimos locales.

Conceptos de Computación Evolutiva



Se denomina **esquema** a un patrón de valores de genes de un cromosoma que puede incluir estados 'no importa' (*don't care*).



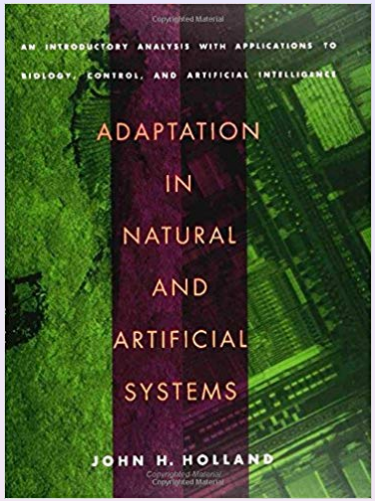
Usando un alfabeto binario, los esquemas se forman del alfabeto $\{0, 1, *\}$.

Por ejemplo, el cromosoma **0110** es una instancia del esquema ***1*0** (donde * significa 'no importa').

Técnicas de Representación

- Cadenas Binarias (Tradicional)
- Códigos de Gray (Binaria)
- Punto Flotante (Binaria)
- Punto Flotante (Real)
- Punto Flotante (Entera)
- Expresiones S en LISP (Programación Genética)
- Listas Binarias de Longitud Variable (*messy-GA*)
- Híbridos (AG Estructurado)
- Otras (matrices, grafos, gramáticas, etc.)

Cadenas Binarias



- Sugerida por Holland en su libro.
- Además de ser una representación "universal" para cualquier tipo de alfabetos, Holland justificó el uso de representación binaria con un argumento teórico.

John H. Holland, **Adaptation in Natural and Artificial Systems. An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence**, MIT Press, Cambridge, Massachusetts, Second Edition, 1992.

Cadenas Binarias

Holland favorece el uso de muchos genes con pocos alelos posibles en vez de contar con pocos genes con muchos alelos posibles.

Teóricamente, la representación binaria favorece la diversidad y la formación de buenos bloques constructores.

La representación binaria es también “natural” para las computadoras.

Cadenas Binarias

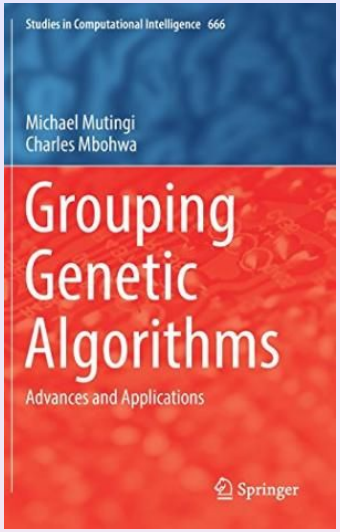
Según Holland, la representación binaria tiene también una justificación biológica: en los cromosomas naturales, es más usual que hayan muchas posiciones y pocos alelos por posición que pocas posiciones y muchos alelos por posición.

Cadenas Binarias

Pese a sus ventajas, las cadenas binarias presentan algunos problemas:

- Escalabilidad
- Epístasis
- Representación natural
- Soluciones ilegales

Códigos de Gray



Motivación: Problemas con el mapeo entre genotipo y fenotipo al usar representación binaria.

Ejemplo:
5 (decimal) = 101
6 (decimal) = 110

Puede verse que hay una diferencia de 1 en el espacio fenotípico, y de 2 en el genotípico (distancia de Hamming).

Códigos de Gray

A este fenómeno se le conoce como **risco de Hamming** (*Hamming cliff*).

Este fenómeno es el que ha propiciado propuestas de representaciones que mantengan la propiedad de adyacencia entre 2 valores consecutivos. De entre ellas, los denominados **códigos de Gray** se cuentan entre las más populares.

Darrell Whitley, "**A free lunch proof for gray versus binary encodings**", in W. Banzhaf et al. (Eds.), *Proceedings of the 1999 Genetic and Evolutionary Computation Conference (GECCO'99)*, Vol. 1, pp. 726–733, Morgan Kaufmann Publishers, San Francisco, California, USA, July 1999.

Códigos de Gray

Decimal	Binario	Código de Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111

Códigos de Gray

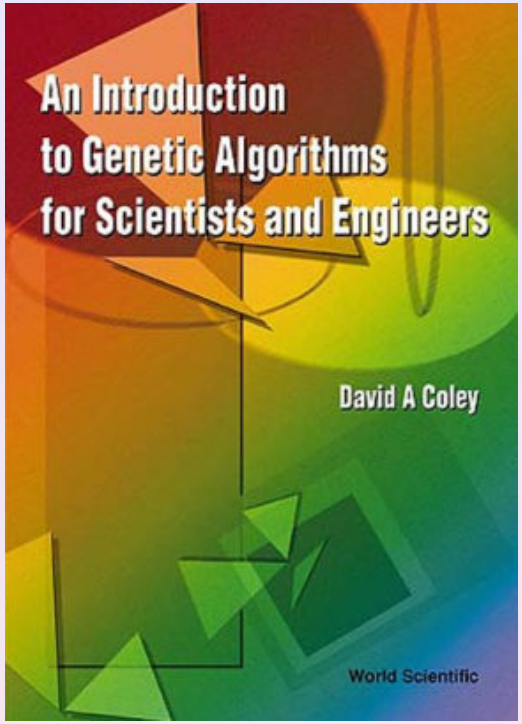
Se presupone que: **b** $=< b_1, ..., b_m >$ es un número binario, y **g** $=< g_1, \ldots, g_m >$ es un número de Gray.

procedure Binario-a-Gray
begin
    $g_1 = b_1$
    for $k = 2$ **to** m **do**
        $g_k = b_{k-1}$ **XOR** b_k
end

Códigos de Gray

procedure Gray-a-Binario
begin
    valor = g_1
    b_1 = valor
    for $k = 2$ **to** m **do**
    begin
        if $g_k = 1$ **then** valor = **NOT** valor
        b_k = valor
    end
end

Representación de Punto Flotante



- Binaria
- IEEE
- Real
- Entera

Uso de Cadenas Binarias

Podemos discretizar el rango de cada variable fijando una precisión deseable y posteriormente se tratan esos valores como si fueran enteros.

Ejemplo:

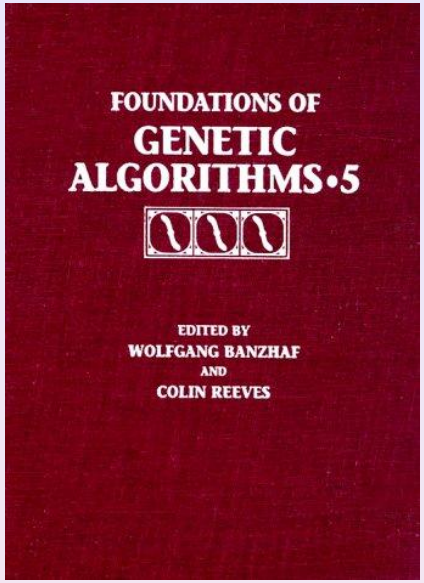
$$-1.5 \leq x \leq 2.0$$

Precisión: 3 dígitos

Tamaño = (int) $[log_2[(l_{sup} - l_{inf}) \times 10^{\text{precisión}}] + 0.99]$

Tamaño = 12 bits

Representación de Punto Flotante



Uso de Cadenas Binarias

Este tipo de codificación presenta ciertas ventajas y desventajas:

- Representación compacta (V)
- Decodificación eficiente (V)
- Problemas ante alta dimensionalidad (D)
- Precisión limitada (D)

Representación de Punto Flotante

Notación Estándar del IEEE

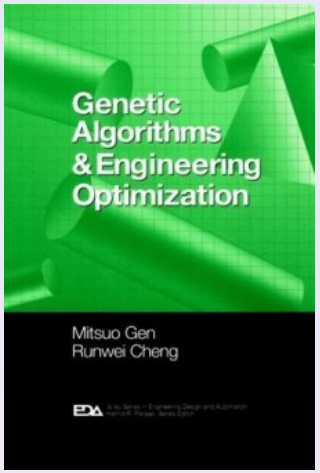
Podemos usar cualquier formato estándar del IEEE para números de punto flotante.

Por ejemplo:

Signo	Exponente	Mantisa
0	1 0 0 0 1 0 1 1	0 1 0 0 . . . 0
1 bit	8 bits	23 bits

Notación en exceso -127
Rango numérico: 2^{-126} a 2^{126}

Representación de Punto Flotante

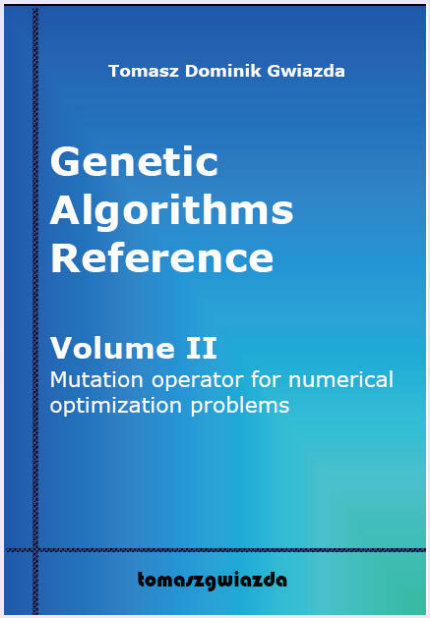


Notación Estándar del IEEE

Este tipo de codificación presenta ciertas ventajas y desventajas:

- Existen notaciones definidas para varias precisiones numéricas (V)
- Decodificación costosa (D)
- Mapeo muy complejo entre el genotipo y el fenotipo (D)
- Muy susceptible a cambios pequeños en algunos campos (p.ej. el exponente) y poco susceptible a cambios mayores en otros (p.ej. la mantisa) (D)

Representación de Punto Flotante



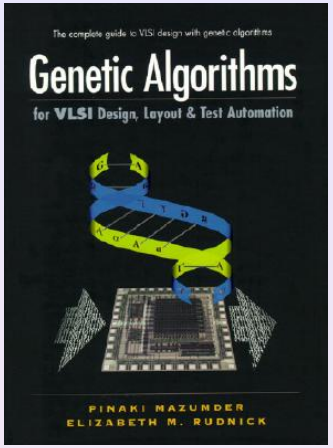
Representación Real

Podemos usar directamente números reales en cada gene:

1.54	-0.29	7.43	-2.15

Realmente estaríamos operando a nivel fenotípico.

Representación de Punto Flotante

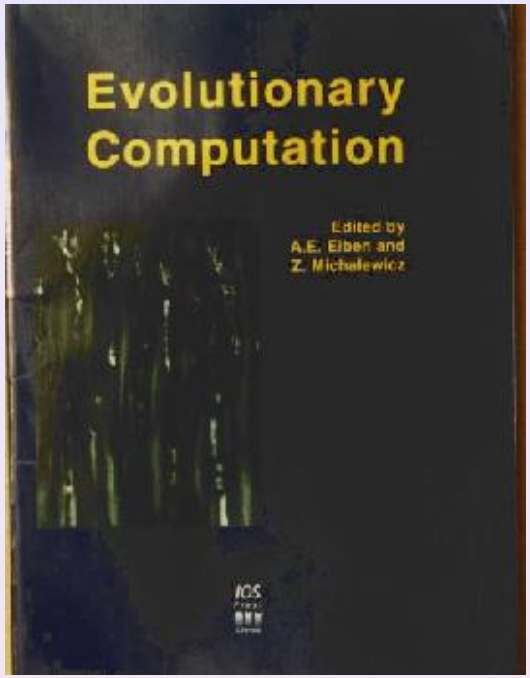


Representación Real

Este tipo de codificación presenta ciertas ventajas y desventajas:

- Requiere de operadores genéticos especiales (D)
- Es una notación más compacta para codificar números reales (V)
- Los teóricos argumentan que las cardinalidades más bajas son las mejores (D)
- Existe evidencia empírica de la efectividad de este tipo de codificación (V)

Representación de Punto Flotante



Representación Entera

Podemos usar:

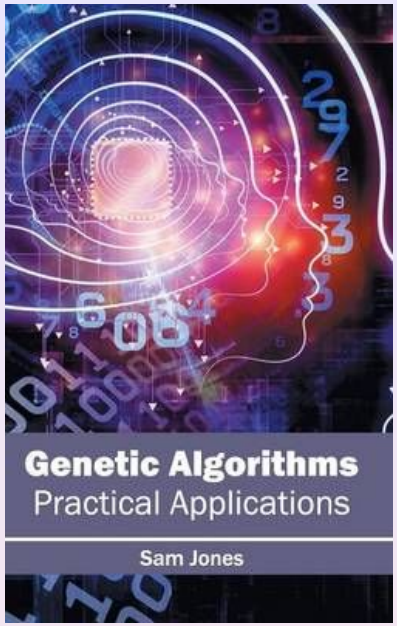
7	2	9	3	2	0	1

= 72.93201

o algo como:

72945	12902	13001	12000

Representación de Punto Flotante

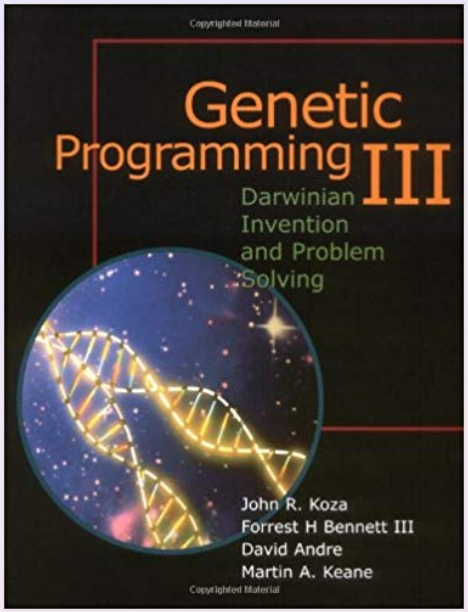


Representación Real

Este tipo de codificación presenta ciertas ventajas y desventajas:

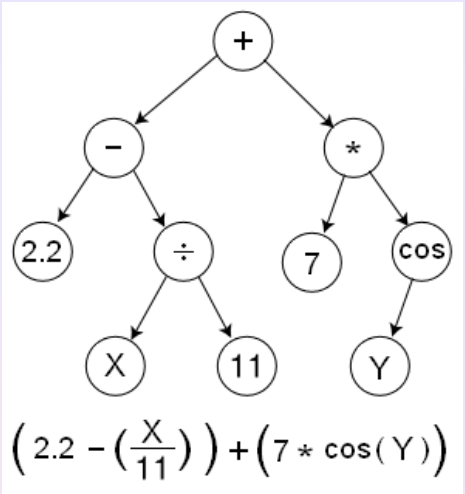
- Limitada por la precisión (D)
- Puede requerir operadores especiales (D)
- Compromiso entre la representación real y la binaria (V)
- Puede producir ahorros de memoria (V)

Programación Genética



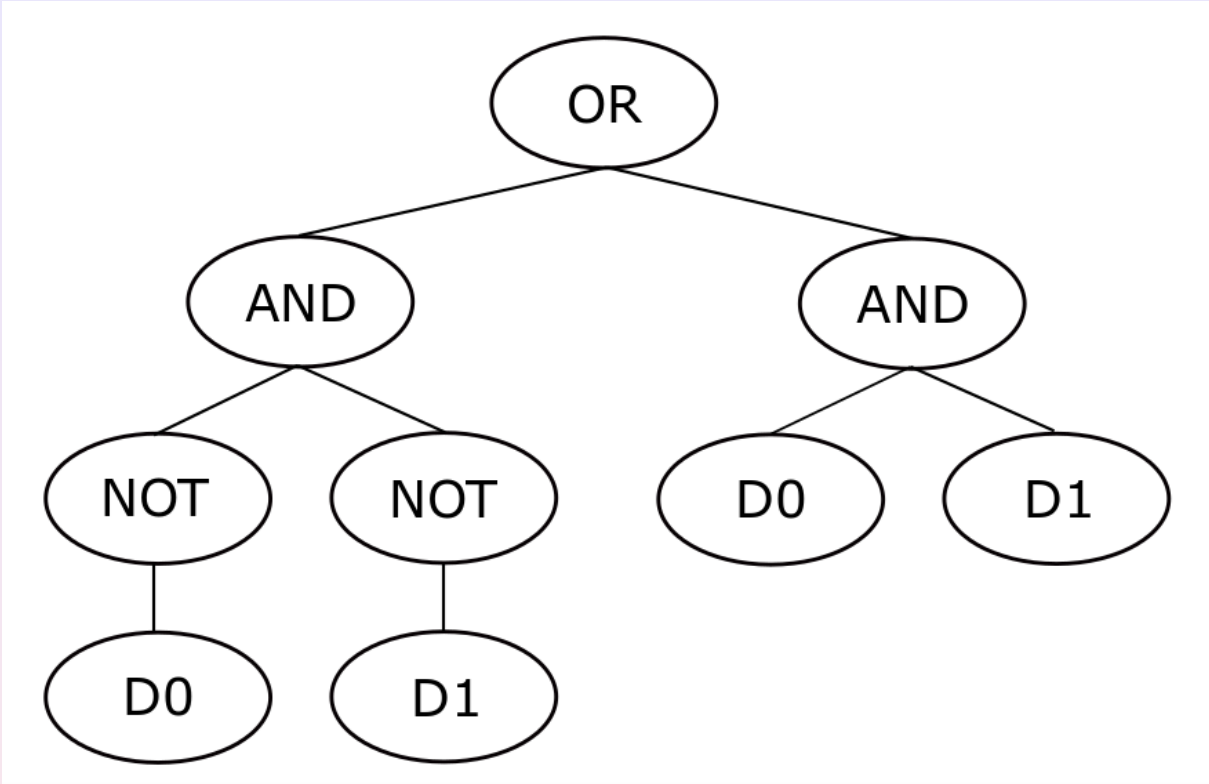
- Uso de árboles para representar programas de computadora
- Se predetermina la máxima profundidad de los árboles, pero no su topología precisa
- El tamaño, forma y contenido de los árboles puede cambiar dinámicamente durante el proceso evolutivo

Programación Genética



Típicamente, se adoptan las siguientes funciones:

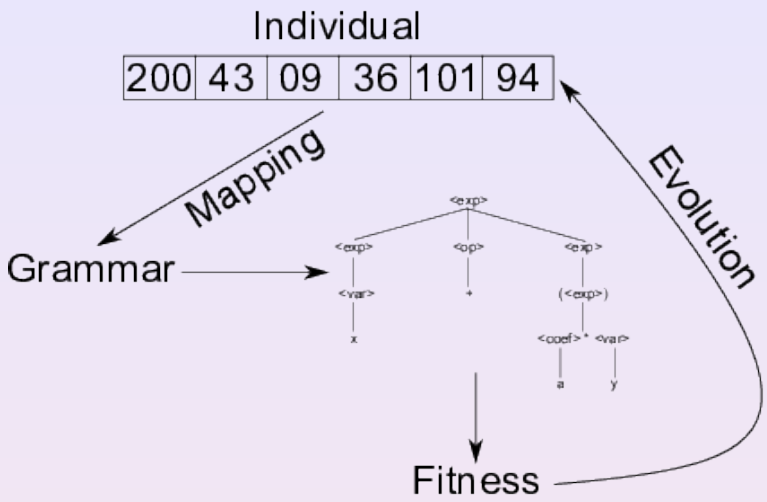
1) Operaciones aritméticas ($+, -, *$, *etc.*)
2) Funciones matemáticas (seno, coseno, exp, log)
3) Operaciones Booleanas (AND, OR, NOT)
4) Operadores condicionales (IF-THEN-ELSE)
5) Funciones que causan iteraciones (DO-UNTIL)
6) Funciones que causan recursión
7) Cualquier función específica del dominio definido



La expresión S equivalente es:

(OR (AND (NOT D0)(NOT D1))(AND (D0 D1))

Programación Genética

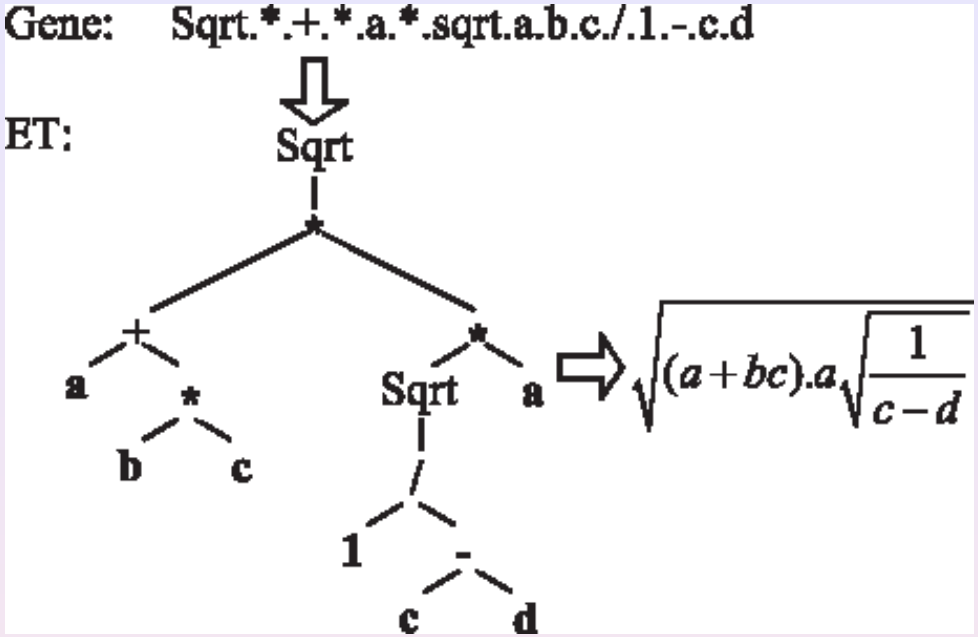


Evolución gramatical (*grammatical evolution*) es una variante de la programación genética en la que se usan cadenas de enteros que se mapean a un programa usando una gramática.

Esta técnica se diseñó con el objetivo de contar con una mayor portabilidad a través de diferentes lenguajes de programación.

Conor Ryan and Michael O'Neill, "**Grammatical Evolution: A Steady State approach**", in *Proceedings of the Second International Workshop on Frontiers in Evolutionary Algorithms*, pp. 419–423, 1998.

Programación Genética



Existe otra variante de la programación genética conocida como **Gene Expression Programming**.

En este caso, se representan programas usando cromosomas lineales de longitud fija y cada cromosoma codifica siempre una expresión válida.

Candida Ferreira, "**Gene Expression Programming: A New Adaptive Algorithm for Solving Problems**", *Complex Systems*, Vol. 13, No. 2, pp. 87–129, 2001.

Algoritmos Genéticos Desordenados

- Desarrollados por Deb y Goldberg hacia fines de los 1980s.
- **Motivación:** resolver problemas "deceptivos" con un algoritmo genético.
- **Características:** longitud cromosómica y tamaño de población variables.
- Se le llama **algoritmo genético desordenado**, porque se contrapone al algoritmo genético "ordenado" (*neat*) que tiene longitud y tamaño de población fijos.

David E. Goldberg, Bradley Korb and Kalyanmoy Deb, "**Messy Genetic Algorithms: Motivation, Analysis and First Results**", *Complex Systems*, Vol. 3, pp. 493–530, 1989.

Algoritmos Genéticos Desordenados

- **Idea fundamental:** representar esquemas en vez de cromosomas.
- **Funcionamiento:** Comenzar con cromosomas cortos, identificar un conjunto de buenos bloques constructores y después incrementar la longitud del cromosoma para propagar estos buenos bloques constructores a lo largo del resto de la cadena.

Algoritmos Genéticos Desordenados

Ejemplos de cadenas válidas:

	(2,1)	(2,0)	(3,0)	(3,1)
(1,1)	(1,0)	(1,1)	(4,1)	(4,0)

El número de la izquierda es la posición cromosómica (*locus*) y el de la derecha es el valor del bit.

Algoritmos Genéticos Desordenados

Peculiaridades de la representación:

- Algunas posiciones pueden ser asignadas a más de un bit **(sobre-especificación)**
- Otras posiciones pueden no ser asignadas a ningún bit **(sub-especificación)**

¿Cómo lidiar con la **sobre-especificación**?

Imponer un orden determinístico de evaluación. Por ejemplo, de izquierda a derecha. De esta manera, se ignoran los valores posteriores al primero detectado para una cierta posición cromosómica.

Ejemplo: si tenemos la cadena

$$\{ (1,0)\ (2,0)\ (4,1)\ (4,0) \}$$

tomamos el valor de 1 para el cuarto bit e ignoramos los demás.

Algoritmos Genéticos Desordenados

¿Cómo lidiar con la **sub-especificación**?

- Tenemos que evaluar la aptitud de cadenas parcialmente definidas.
- Podemos ver una cadena como un esquema candidado.

Ejemplo: si tenemos la cadena

$$\{ (1,0)\ (2,0)\ (4,1)\ (4,0) \}$$

se trata del esquema candidato: $00*1$.

Algoritmos Genéticos Desordenados

¿Cómo lidiar con la **sub-especificación**?

Hay dos métodos principales:

(1) El método de los promedios

(2) El método de las plantillas competitivas

Algoritmos Genéticos Desordenados

Método de los Promedios

- Generar aleatoriamente valores para los lugares faltantes un cierto número (pre-definido) de veces.
- La aptitud de la cadena sub-especificada será el promedio de aptitud de estas muestras.

Algoritmos Genéticos Desordenados

Método de los Promedios

- **Motivación:** se intenta calcular el promedio del esquema candidato.
- **Problema:** La varianza de esta aptitud promedio frecuentemente será demasiado alta para que se pueda obtener un valor significativo mediante un muestreo aleatorio.

Algoritmos Genéticos Desordenados

Método de las Plantillas Competitivas

- Usar un *hillclimber* para obtener un óptimo local.
- Al ejecutar el messy GA, se evalúan las cadenas sub-especificadas llenando los bits faltantes con el óptimo local. De esa manera, es posible calcular la aptitud de cualquier esquema candidato.

Algoritmos Genéticos Desordenados

Método de las Plantillas Competitivas

- **Motivación:** Por definición, un óptimo local es una cadena que no puede mejorarse con el cambio de un solo bit. Por tanto, si los bits definidos de un esquema candidato mejoran el óptimo local, vale la pena efectuar una mayor exploración.
- **Problema:** El método es víctima de la explosión combinatoria.

Algoritmos Genéticos Desordenados

Fases de Operación

Los algoritmos genéticos desordenados operan en 2 fases:

- Fase Primordial
- Fase Yuxtaposicional

Algoritmos Genéticos Desordenados

Fase Primordial

El objetivo de la **fase primordial** es generar esquemas cortos que sirven como los bloques constructores en la **fase yuxtaposicional** en que éstos se combinan.

Problema Fundamental: ¿Cómo decidir qué tan largos deben ser estos "**esquemas cortos**"?

Algoritmos Genéticos Desordenados

Fase Primordial

La solución que plantearon Goldberg y sus colegas al problema de la longitud de los esquemas cortos fue adivinar el "orden" de dichos esquemas.

El **orden** de un esquema es el número de posiciones fijas que tiene. Por ejemplo: o(0**10) = 3.

Fase Primordial

Si se adivina el orden del esquema (al que llamaremos k), entonces se procede a generar por enumeración todos los esquemas de ese orden y de la longitud requerida, l.

Por ejemplo, si $l = 8$ y $k = 3$, tenemos:

$$\{(1, 0)\ (2, 0)\ (3, 0)\}$$
$$\{(1, 0)\ (2, 0)\ (3, 1)\}$$
$$\vdots$$
$$\vdots$$
$$\{(1, 1)\ (2, 1)\ (3, 1)\}$$
$$\{(1, 0)\ (2, 0)\ (4, 0)\}$$
$$\{(1, 0)\ (2, 0)\ (4, 1)\}$$
$$\vdots$$
$$\vdots$$
$$\{(6, 1)\ (7, 1)\ (8, 1)\}$$

Algoritmos Genéticos Desordenados

Fase Primordial

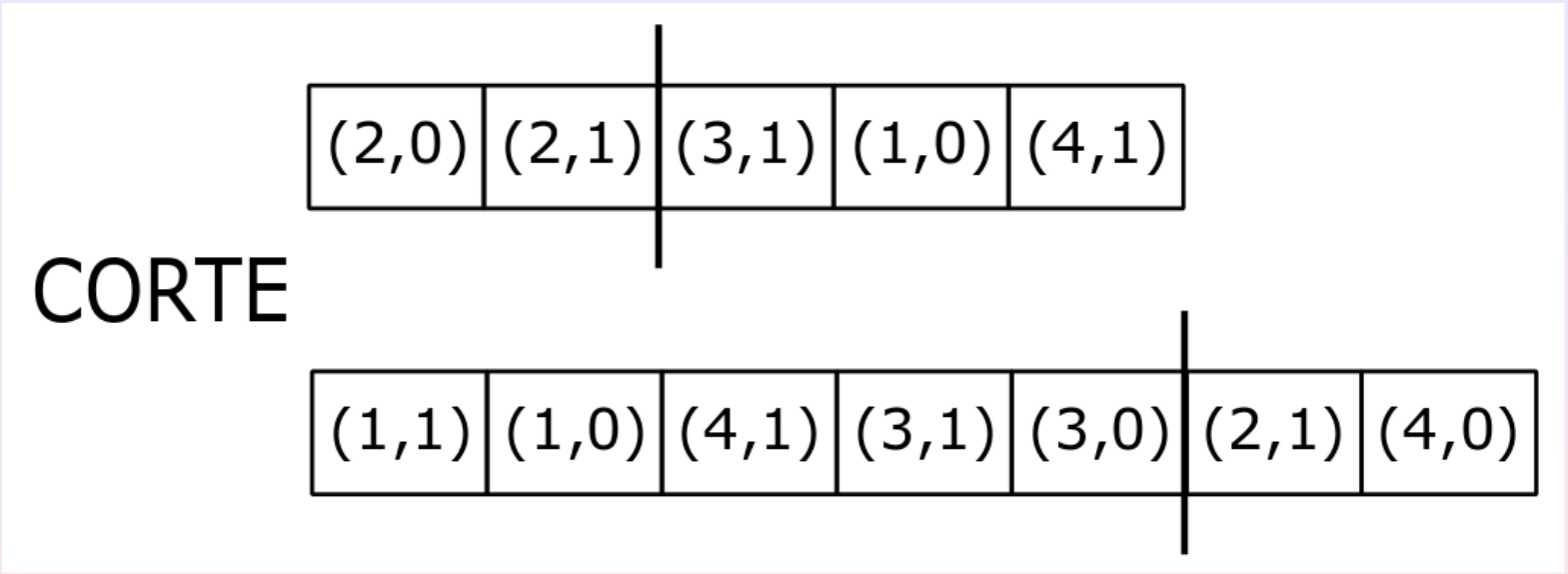
Estos esquemas cortos (de orden predefinido) se generan en la fase primordial y luego evaluamos sus aptitudes. Después de eso aplicamos sólo selección a la población (sin cruza ni mutación) para propagar los buenos bloques constructores, y se borra la mitad de la población a intervalos regulares.

Fase Yuxtaposicional

Después de un cierto número (predefinido) de generaciones, entramos a la fase yuxtaposicional.

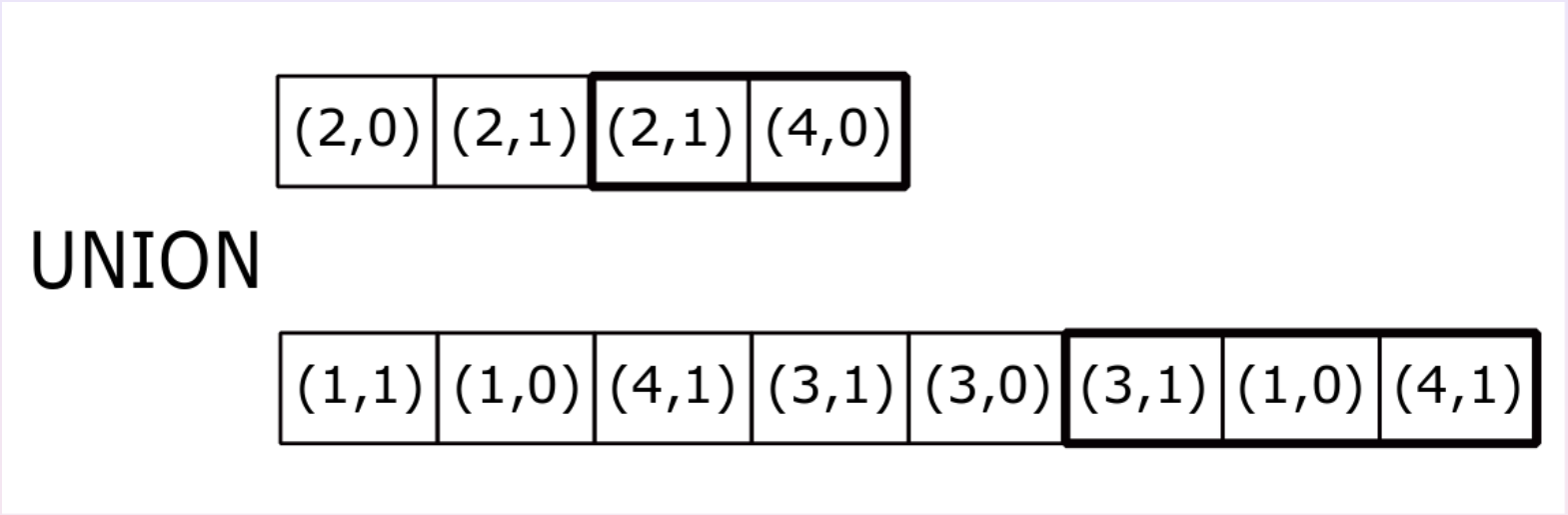
A partir de este punto, el tamaño de la población permanecerá fijo y usaremos selección y 2 operadores llamados **corte** (*cut*) y **unión** (*splice*).

Debido a la naturaleza de este algoritmo, estos operadores siempre producen cadenas válidas.



Fase Yuxtaposicional

Operador de Corte.



Fase Yuxtaposicional

Operador de Unión.

Algoritmos Genéticos Desordenados

El algoritmo genético desordenado está diseñado para resolver problemas sumamente difíciles para un algoritmo genético tradicional (los problemas deceptivos).

A pesar de ser una propuesta interesante, ha tenido poco uso práctico porque suele ser víctima de la "maldición de la dimensionalidad".

Adicionalmente, el hecho de que no se hayan podido encontrar problemas deceptivos del mundo real, ha condenado a este algoritmo a la obsolescencia.

Algoritmo Genético Estructurado

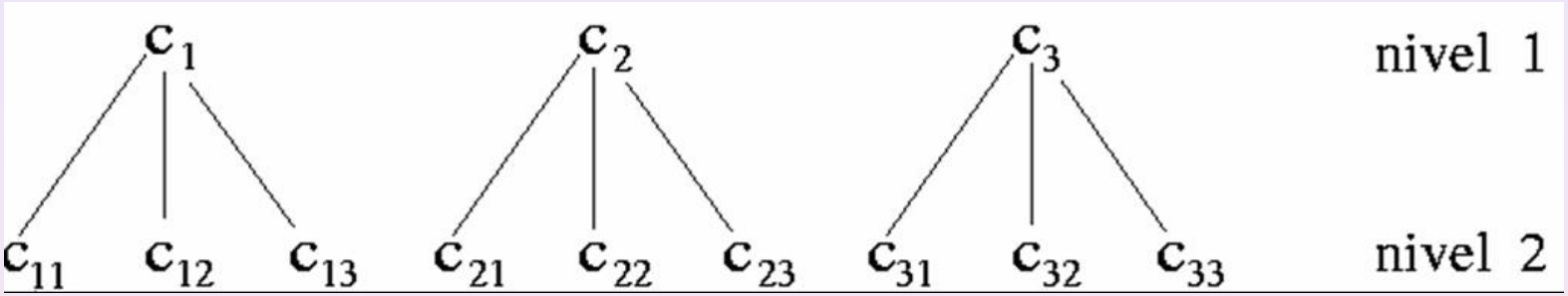
Representación híbrida propuesta por Dipankar Dasgupta a principios de los 1990s.

Es un compromiso entre la representación lineal del algoritmo genético tradicional y la de árbol usada por la programación genética.

Usa una representación jerárquica con un mecanismo de dominancia similar al de los diploides.

Dipankar Dasgupta and Douglas R. McGregor, "**Nonstationary Function Optimization using the Structured Genetic Algorithm**", in *Proceedings of Parallel Problem Solving from Nature (PPSN 2)*, pp. 145–154, Springer-Verlag, Brussels, Belgium, September 1992.

Algoritmo Genético Estructurado



El algoritmo genético estructurado codifica estructuras genéticas de varios niveles (grafos dirigidos o árboles) como se muestra en la imagen de arriba.

Algoritmo Genético Estructurado

Los genes en cualquier nivel pueden ser **pasivos** o **activos**, pero los genes de alto nivel activan o desactivan conjuntos de genes de más bajo nivel, lo que significa que cualquier cambio pequeño en un alto nivel se magnifica en los niveles inferiores.

Algoritmo Genético Estructurado

La idea principal del algoritmo genético estructurado es que los genes de alto nivel deben **explorar** las áreas potenciales del espacio de búsqueda y los de bajo nivel deben **explotar** ese sub-espacio.

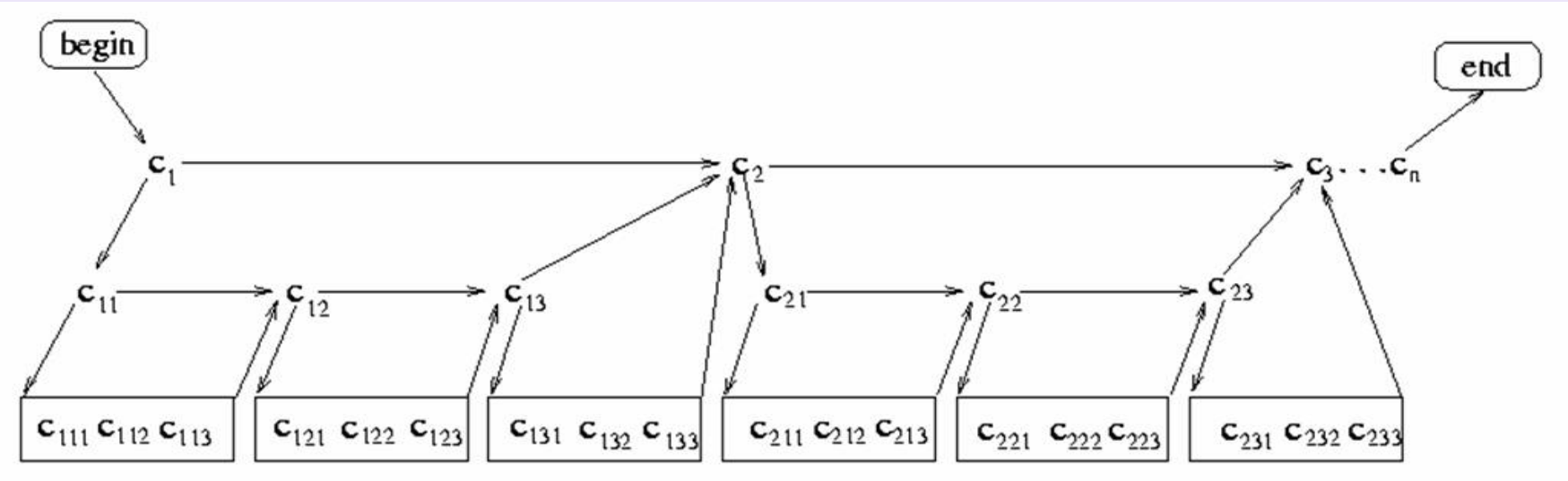
El uso principal de este tipo de técnica es para funciones dinámicas.

Algoritmo Genético Estructurado

A pesar de tener una estructura jerárquica, en un algoritmo genético estructurado se codifica como un cromosoma lineal de longitud fija:

$$(c_1 \quad c_2 \quad c_3 \quad c_{11} \quad c_{12} \quad c_{13} \quad c_{21} \quad c_{22} \quad c_{23} \quad c_{31} \quad c_{32} \quad c_{33})$$

Algoritmo Genético Estructurado



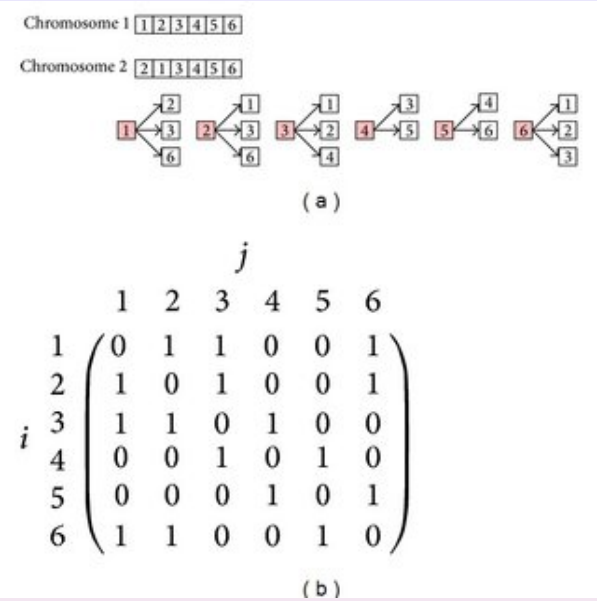
Sin embargo, el algoritmo genético estructurado requiere de una estructura de datos más complicada que la del algoritmo genético tradicional.

Algoritmo Genético Estructurado

Características

- Mostró ser útil para resolver problemas dinámicos (aquellos donde la posición del óptimo cambia en el tiempo).
- Su implementación es bastante más complicada que la de un algoritmo genético tradicional.
- El mapeo entre el genotipo y el fenotipo es extremadamente complejo.
- Su desempeño se degrada rápidamente conforme se aumenta el número de jerarquías utilizadas.

Otras Representaciones



Existen muchas otras representaciones posibles para los algoritmos genéticos. Por ejemplo:

- Grafos
- Matrices
- Gramáticas
- Pilas

Recomendaciones para el diseño de buenas representaciones

Palmer [1994] hizo varias recomendaciones en el contexto de representaciones de árbol, en torno los puntos clave a tomarse en cuenta al proponer una representación:

1. Nuestra codificación debe poder representar todos los fenotipos posibles.

Charles Chamber Palmer, "**An approach to a problem in network design using genetic algorithms**", PhD thesis, Polytechnic University, Troy, New York, USA, 1994.

Recomendaciones para el diseño de buenas representaciones

2. Nuestra codificación no debe tener sesgos, de manera que todos los individuos posibles se encuentren representados de manera equitativa en el conjunto de todos los genotipos posibles.
3. Nuestra codificación no debería permitir soluciones infactibles.

Recomendaciones para el diseño de buenas representaciones

4. La decodificación del genotipo al fenotipo debiera ser simple.
5. Una codificación debe poseer localidad (o sea, cambios pequeños en el genotipo debieran resultar en cambios pequeños en el fenotipo).

Recomendaciones para el diseño de buenas representaciones

Ronald [1995] hizo también sugerencias en torno a cómo elegir una buena representación:

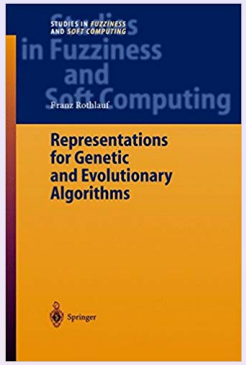
1. Las codificaciones deben ajustarse a un conjunto de operadores genéticos de tal forma que se preserven (o transmitan) los bloques constructores de padres a hijos.

S.P. Ronald, "**Genetic algorithms and permutation-encoded problems: Diversity preservation and a study of multimodality**", PhD thesis, The University of South Australia, 1995.

Recomendaciones para el diseño de buenas representaciones

2. Una buena codificación debe minimizar la epístasis.
3. Deben preferirse las soluciones factibles.
4. El problema debe estar representado a un nivel correcto de abstracción.
5. Las codificaciones deben explotar un mapeo apropiado del genotipo al fenotipo en caso de que no sea posible producir un mapeo simple entre estos 2 espacios.

Recomendaciones para el diseño de buenas representaciones



6. No debieran usarse formas isomórficas, en las cuales el fenotipo de un individuo es codificado con más de un genotipo (Rothlauf [2002] mostró que las formas isomórficas mejoran el desempeño de un algoritmo evolutivo).

Franz Rothlauf, **Representations for Genetic and Evolutionary Algorithms**, Physica-Verlag, Heidelberg, Germany, 2002, ISBN 3-7908-1496-2.

Técnicas de Selección

- Selección Proporcional
- Selección mediante torneo
- Selección de Estado Uniforme

Técnicas de Selección

Selección Proporcional

Este nombre describe a un grupo de esquemas de selección originalmente propuestos por Holland que eligen individuos de acuerdo a su contribución de aptitud con respecto al total de la población.

Técnicas de Selección

Técnicas de Selección Proporcional

- La Ruleta
- Sobrante Estocástico
- Universal Estocástica
- Muestreo Determinístico

Técnicas de Selección

Técnicas de Selección Proporcional

Existen también mecanismos complementarios (o aditamentos) a las técnicas de selección proporcional:

- Escalamiento Sigma
- Jerarquías
- Selección de Boltzmann

La Ruleta

Fue propuesta por Kenneth De Jong [1975] en su tesis doctoral.

Fue el método más comúnmente usado en los orígenes de los algoritmos genéticos.

El algoritmo de esta técnica es simple, pero ineficiente. Su complejidad es $\mathcal{O}(N^2)$.

Principal problema: El individuo menos apto puede ser seleccionado más de una vez, desviándose así del valor esperado.

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.

La Ruleta: Implementación de De Jong

1. Calcular la suma de valores esperados T.
2. Repetir N veces (N es el tamaño de la población):
   1. Generar un número aleatorio **r** entre 0.0 y T.
   2. Ciclar a través de los individuos de la población sumando los valores esperados hasta que la suma sea mayor o igual a **r**.
   3. El individuo que haga que esta suma exceda el límite es el seleccionado.

Técnicas de Selección

Sobrante Estocástico

Esta técnica fue propuesta por Brindle [1981] y Booker [1982] con el objetivo de acercarse más a los valores esperados de cada individuo.

La idea fundamental de este método es asignar de manera determinista las partes enteras de los valores esperados para cada individuo.

A. Brindle, "**Genetic Algorithms for Function Optimization**", PhD thesis, Department of Computer Science, University of Alberta, Edmonton, Alberta, Canada, 1981.

Lashon B. Booker, "**Intelligent Behavior as an Adaptation to the Task Environment**", PhD thesis, Logic of Computers Group, University of Michigan, Ann Arbor, Michigan, USA, 1982.

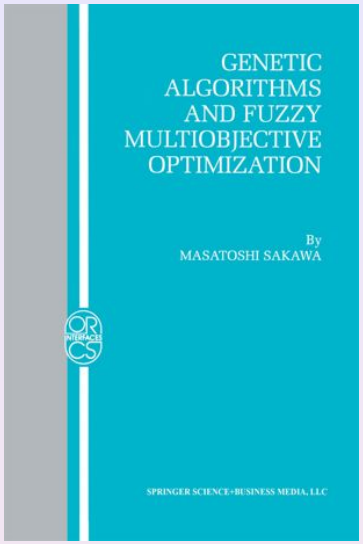
Principal problema: Aunque reduce las diferencias con respecto a los valores esperados, puede causar convergencia prematura.

Sobrante Estocástico

Hay 2 variantes de esta técnica que difieren en la forma en que se manejan las partes decimales de los valores esperados:

1. **Sin Reemplazo**: Usar *flip* con las partes decimales para elegir los padres restantes.
2. **Con Reemplazo**: Construir una ruleta con las partes decimales y usarla para seleccionar los padres faltantes.

Técnicas de Selección



Sobrante Estocástico (Complejidad Algorítmica)

- La versión con reemplazo es $\mathcal{O}(n^2)$
- La versión sin reemplazo es $\mathcal{O}(n)$

La más popular es la versión sin reemplazo, la cual parece ser superior a la ruleta [Booker, 1982].