# Introducción a la Computación Evolutiva

Carlos A. Coello Coello

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

**Clase 12**

# Las Funciones de la Carretera Real

El Teorema de los Esquemas, por sí mismo, no hace referencia a los efectos positivos de la selección (asignar muestras cada vez mayores de los esquemas que han mostrado tener un mejor desempeño), sino únicamente a los aspectos de perturbación de la cruza.

Tampoco aborda la pregunta de cómo hace la cruza para recombinar los esquemas más aptos, aunque ésta parece ser la mayor fuente de poder del algoritmo genético.

La hipótesis de los bloques constructores dice que la cruza combina esquemas cortos y de alto desempeño demostrado para formar candidatos más aptos, pero no da una descripción detallada de cómo ocurre esta combinación.

# Las Funciones de la Carretera Real

Para investigar el procesamiento de esquemas y la recombinación en más detalle, Stephanie Forrest y Melanie Mitchell [1993] diseñaron ciertos paisajes de aptitud llamados de la **Carretera Real** (*Royal Road*), que intentaron capturar la esencia de los bloques constructores en una forma idealizada.

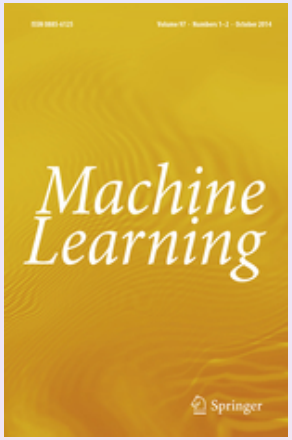
Stephanie Forrest and Melanie Mitchell, "**What makes a Problem Hard for a Genetic Algorithm? Some Anomalous Results and Their Explanation**", *Machine Learning*, Vol. 13, pp. 285–319, 1993.

# Las Funciones de la Carretera Real

La Hipótesis de los Bloques Constructores sugiere 2 características de los paisajes de aptitud que son particularmente relevantes a los algoritmos genéticos:

1. La presencia de esquemas cortos, de bajo orden y altamente aptos, y
2. La presencia de “escalones” intermedios (esquemas de orden intermedio de aptitud más alta que resultan de combinaciones de los esquemas de orden menor y que, en turno, pueden combinarse para crear esquemas de aptitud aún mayor).

# Las Funciones de la Carretera Real




Las funciones que diseñaron Forrest y Mitchell [1993] se esperaba que (siguiendo la hipótesis de los bloques constructores) tenderían una "carretera real" al algoritmo genético de manera que pudiera llegar fácilmente a la cadena óptima.

Asimismo, la hipótesis era que las técnicas escalando la colina tendrían dificultades con estas funciones, porque se necesitaba optimizar un gran número de posiciones de la cadena simultáneamente para moverse de una instancia de bajo orden a una de orden intermedio o alto.

# Las Funciones de la Carretera Real

Se utilizaron funciones del tipo siguiente:

$$R_1(x) = \sum_{i=1}^{8} c_i \delta_i(x) \tag{1}$$

$$\delta_i(x) = \begin{cases} 1 & \text{si } x \in S_i \\ 0 & \text{en caso contrario} \end{cases} \tag{2}$$

Donde: $S_i$ es un conjunto predefinido de esquemas y $c_i$ son coeficientes asignados a los esquemas.

En este caso, $c_i = 8$ para toda $i = 1, \ldots 8$.

# Las Funciones de la Carretera Real

En los experimentos se utilizó selección proporcional con truncamiento sigma.

El número esperado de copias de un individuo con aptitud $F_i$ es:

$$\frac{F_i - \overline{F}}{2\sigma} + 1 \tag{3}$$

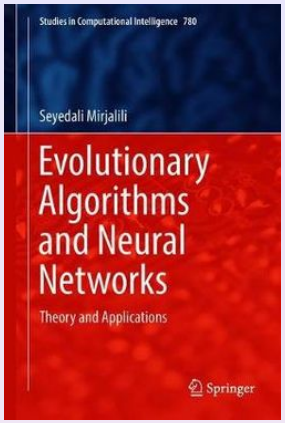
Donde $\overline{F}$ denota el valor promedio de aptitud en la población y $\sigma$ su desviación estándar.

Se usó un porcentaje de cruza de 0.7 y uno de mutación de 0.005.

# Las Funciones de la Carretera Real

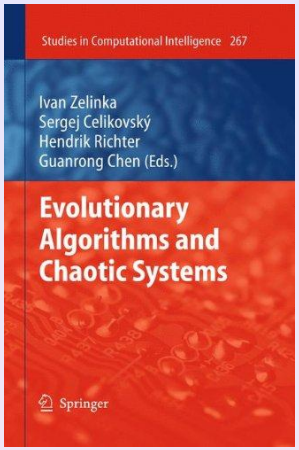
Para los experimentos se usaron tres tipos de *hillclimbers*:

1. **Steepest-Ascent Hill Climbing (SAHC)**:
   1. Elegir aleatoriamente una cadena del dominio de búsqueda. Llamarla la cima actual.
   2. Recorrer la cadena de izquierda a derecha invirtiendo cada bit, evaluando y recordando la aptitud de la cadena resultante.
   3. Si alguna de las cadenas generadas posee una mejor aptitud que las obtenidas hasta ese momento, llamarla la cima actual.
   4. Si no hay incrementos en la aptitud, almacenar la cima actual e ir al paso 1; si no, ir al paso 2 con la nueva cima actual.
   5. Cuando se hayan efectuado un total de $E$ evaluaciones, regresar la cima actual encontrada hasta ahora.



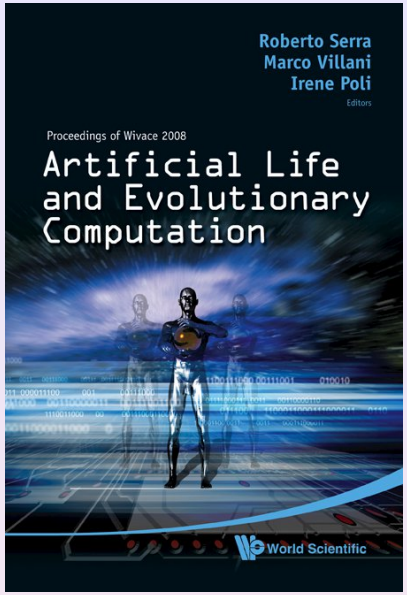

2. **Next-Ascent Hill Climbing (NAHC)**:
   1. Elegir aleatoriamente una cadena del dominio de búsqueda. Llamarla la cima actual.
   2. Para cada uno de los bits de la cadena: invertir el bit. Si la cadena resultante tiene una aptitud mayor que la cima actual, etiquetarla como la nueva cima actual. Si no, regresar el bit a su valor original.
   3. Si no se encontró incremento en la aptitud, almacenar la cima actual e ir al paso 1.
   4. Regresar la cima actual después de $E$ evaluaciones.

# Las Funciones de la Carretera Real




3. **Random Mutation Hill Climbing (RMHC)**:
   1. Elegir aleatoriamente una cadena del dominio de búsqueda. Llamarla la cima actual.
   2. Elegir aleatoriamente una posición de la cadena e invertir el bit en esa posición. Si la cadena resultante posee una aptitud mayor o igual a la de la cima actual, llamarla cima actual.
   3. Ir al paso 2 hasta que haya sido encontrada la cadena de mejor aptitud o hasta que se hayan hecho $E$ evaluaciones.
   4. Regresar la cima actual.

# Las Funciones de la Carretera Real




Aunque estas funciones estaban diseñadas para hacer que un algoritmo genético tuviese un mucho mejor desempeño que los *hillclimbers*, ocurrieron cosas extrañas en los experimentos.

Uno de los *hillclimbers* que usaron (el RHMC) superó por mucho al algoritmo genético (encontró la solución 10 veces más rápido que el algoritmo genético).

# Las Funciones de la Carretera Real

Tras efectuar un análisis de la función en la que el algoritmo genético tuvo problemas, se determinó que una de las causas fueron los **hitchhikers**, que limita seriamente el paralelismo implícito del algoritmo genético, restringiendo los esquemas muestreados en ciertos lugares.

Estos genes parásitos limitan el efecto de la cruza para recombinar bloques constructores, y hacen que converjan hacia esquemas equivocados en diversas particiones.

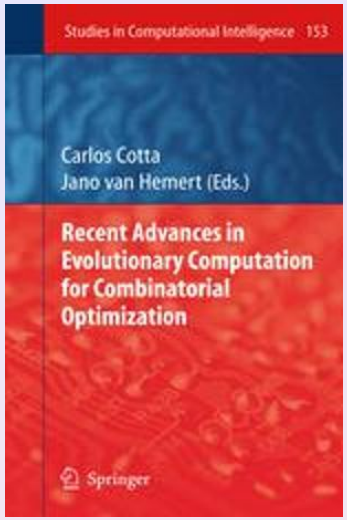
Sin embargo, este fenómeno no debe resultar demasiado sorprendente si se considera que se ha observado en la genética real.

# Las Funciones de la Carretera Real

Para lidiar con este problema se propuso un **Algoritmo Genético Idealizado**, y se concluyeron varias cosas importantes en torno a cómo debe diseñarse un algoritmo genético convencional para evitar el **hitchhiking**:

1. La población tiene que ser suficientemente grande, el proceso de selección debe ser suficientemente lento, y el porcentaje de mutación debe ser suficientemente alto para asegurar que no haya ninguna posición que permanezca fija con un solo valor en ninguna cadena.
2. La selección tiene que ser suficientemente fuerte como para preservar los esquemas deseados que se han descubierto, pero también tiene que ser suficientemente lenta (o, de manera equivalente, la aptitud relativa de los esquemas deseables no traslapados tiene que ser suficientemente pequeña) para prevenir que ocurra algún **hitchhiking** significativo en algunos esquemas altamente aptos que pueda eliminar esquemas deseables de otras partes de la cadena.
3. El porcentaje de cruza tiene que ser tal que el tiempo en que ocurra una cruza que combine dos esquemas deseados sea pequeño con respecto al tiempo de descubrimiento de los esquemas deseados.

# Las Funciones de la Carretera Real




Estos mecanismos no son compatibles entre sí.

Por ejemplo, un alto porcentaje de mutación está en contraposición con una selección fuerte.

Por lo tanto, debe cuidarse de que haya algún equilibrio de estos puntos a la hora de aplicarlos.

# ¿Cuándo debe usarse un algoritmo genético?

Es adecuado si el espacio de búsqueda es muy grande, accidentado, poco comprendido, o si la función de aptitud tiene mucho ruido, y si la tarea no requiere que se encuentre el óptimo global (encontrar una solución bastante buena con cierta rapidez resulta suficiente).

Si el espacio de búsqueda es muy pequeño, entonces el problema se puede resolver mediante búsqueda exhaustiva, y no se justifica el uso de un algoritmo genético.

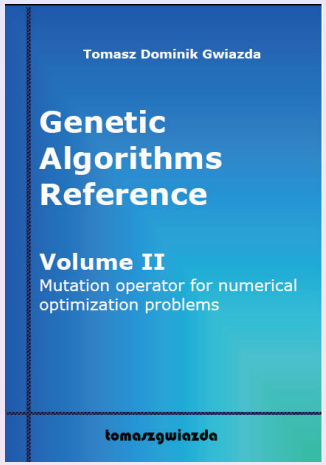
Si el espacio de búsqueda no está accidentado y es unimodal, entonces una técnica de gradiente como "escalando la colina con ascenso empinado" será mucho más eficiente que un algoritmo genético.

# ¿Cuándo debe usarse un algoritmo genético?

Si el espacio de búsqueda se conoce bien (p. ej. alguna instancia pequeña del problema del viajero) es posible diseñar métodos de búsqueda que usen conocimiento específico sobre el dominio para superar fácilmente a una técnica independiente del dominio como el algoritmo genético.

Si la función de aptitud tiene ruido (por ejemplo, si involucra tomar medidas sujetas a error de un proceso del mundo real tal como la visión de un robot), un método de búsqueda que use un solo candidato a la vez (como escalando la colina) será arrastrada inevitablemente por rutas erróneas debido al ruido, mientras que el algoritmo genético tendrá un desempeño razonable porque trabaja mediante la acumulación de estadísticas de aptitud a través de las generaciones.

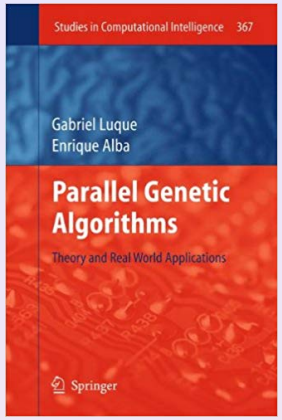
# ¿Cuándo debe usarse un algoritmo genético?




Los consejos anteriores deben tomarse con ciertas precauciones, porque no hay reglas universales sobre cuándo utilizar un algoritmo genético para resolver un problema y cuándo no hacerlo.

Su desempeño normalmente dependerá de detalles tales como el método de codificación de las soluciones candidatas, los operadores, los valores de los parámetros, y el criterio adoptado para medir el éxito del algoritmo.

# Diseño de Funciones Deceptivas




## Problema deceptivo mínimo

La idea básica tras el diseño de funciones deceptivas para un algoritmo genético es violar de manera extrema la hipótesis de los bloques constructores.

En otras palabras, buscaremos ahora que los bloques cortos y de bajo orden nos conduzcan a bloques constructores largos y de mayor orden que sean incorrectos (subóptimos).

# Diseño de Funciones Deceptivas

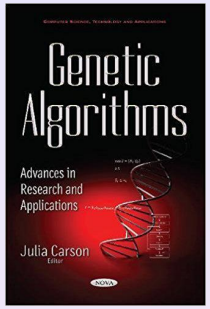
El problema deceptivo más pequeño posible es de dos bits. Su descripción se presenta a continuación.

Supongamos que tenemos un conjunto de cuatro esquemas de orden 2 como se indica a continuación:

| esquema | aptitud |
|---|---|
| $***0******0*$ | $f_{00}$ |
| $***0******1*$ | $f_{01}$ |
| $***1******0*$ | $f_{10}$ |
| $***1******1*$ | $f_{11}$ |

$f_{11}$ es la aptitud máxima posible (óptimo global).
$f_{11} > f_{00}; f_{11} > f_{01}; f_{11} > f_{10}$

# Diseño de Funciones Deceptivas




Ahora procederemos a introducir el elemento deceptivo usando la idea siguiente: buscaremos que en nuestro problema uno de los esquemas subóptimos de orden 1 (o los dos) sea mejor que los esquemas de orden 1 del óptimo.

Para poner el problema en una perspectiva más adecuada, vamos a normalizar todas las aptitudes con respecto al complemento del óptimo global:

$$r = \frac{f_{11}}{f_{00}}; c = \frac{f_{01}}{f_{00}}; c' = \frac{f_{10}}{f_{00}}$$

# Diseño de Funciones Deceptivas

Podemos re-escribir ahora la condición de globalidad en forma normalizada:

$$\frac{f_{11}}{f_{00}} > \frac{f_{00}}{f_{00}}; \frac{f_{11}}{f_{00}} > \frac{f_{01}}{f_{00}}; \frac{f_{11}}{f_{00}} > \frac{f_{10}}{f_{00}}$$

Re-escribamos ahora la condición deceptiva:

$$\frac{f_{00} + f_{01}}{2} > \frac{f_{10} + f_{11}}{2}$$

$$\frac{f_{00} + f_{01}}{f_{00}} > \frac{f_{10} + f_{11}}{f_{00}}$$

$$1 + \frac{f_{01}}{f_{00}} > \frac{f_{10}}{f_{00}} + \frac{f_{11}}{f_{00}}$$

# Diseño de Funciones Deceptivas

$$1 + c > c' + r$$

$$r + c' < c + 1$$

$$r < c + 1 - c'$$

$$f(0*) > f(1*)$$

$$f(*0) > f(*1)$$

# Diseño de Funciones Deceptivas

En estas expresiones estamos ignorando todos los demás alelos de las cadenas cromosómicas que no sean las 2 posiciones definidas antes indicadas y las expresiones anteriores implican un promedio sobre todas las cadenas contenidas en el subconjunto de similitud.

De tal forma que deben cumplirse las siguientes expresiones:

$$\frac{f(00)+f(01)}{2} > \frac{f(10)+f(11)}{2}$$

$$\frac{f(00)+f(10)}{2} > \frac{f(01)+f(11)}{2}$$

Sin embargo, estas 2 expresiones no pueden cumplirse simultáneamente, pues de hacerlo $f_{11}$ no sería el óptimo global.

# Diseño de Funciones Deceptivas

Sin pérdida de generalidad, supondremos que la primera expresión es cierta:

$$f(0*) > f(1*)$$

A partir de lo anterior, podemos concluir que:

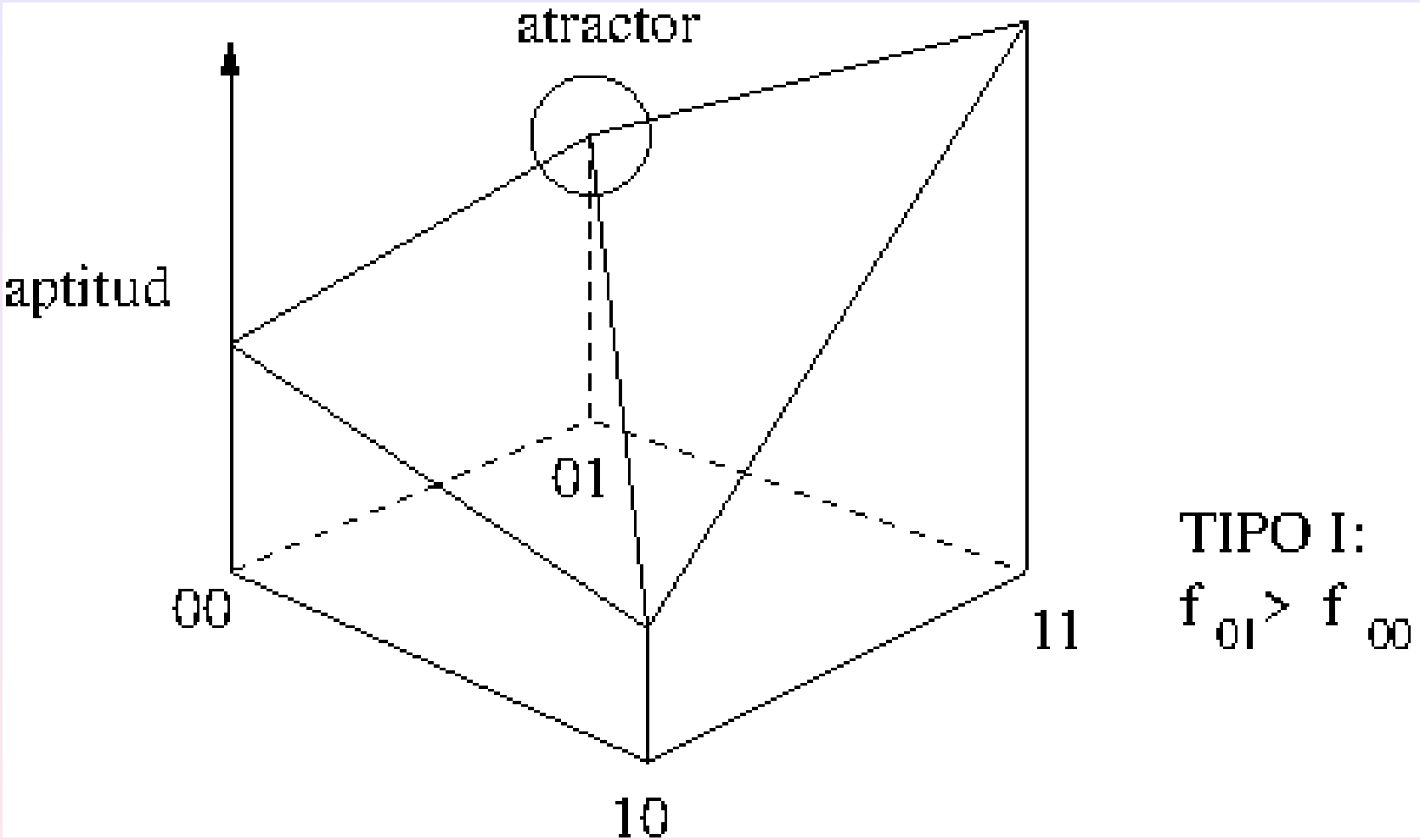
$$c' < 1; c' < c$$

Tenemos entonces 2 clases de problemas deceptivos:

TIPO I: $f_{01} > f_{00}$ $(c > 1)$

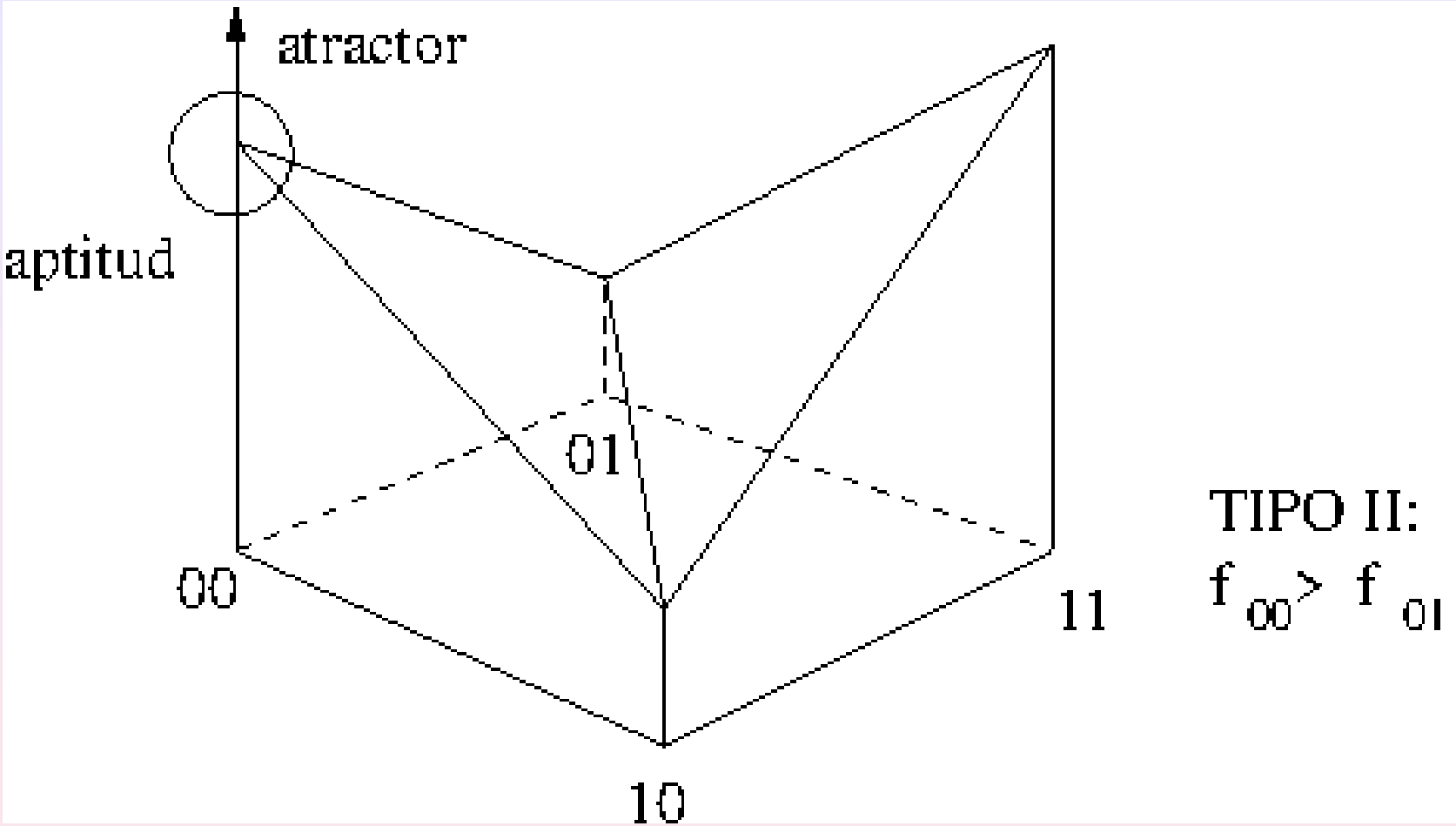
TIPO II: $f_{00} \geq f_{01}$ $(c \leq 1)$

# Diseño de Funciones Deceptivas




Representación gráfica del problema deceptivo tipo I.

# Diseño de Funciones Deceptivas




Representación gráfica del problema deceptivo tipo II.

# Diseño de Funciones Deceptivas

Estos 2 tipos de problemas son deceptivos y puede demostrarse que ninguno de ellos puede expresarse como una combinación lineal de los valores alélicos del individuo.

Ninguno de estos casos puede expresarse como:

$$f(x_1, x_2) = b + \sum_{i=1}^{2} a_i x_i$$

En términos biológicos, tenemos un problema epistático.

Puesto que puede demostrarse que ningún problema de 1 bit puede ser deceptivo, el problema de 2 bits antes indicado es el **problema deceptivo mínimo**.

# Otros Resultados Teóricos Importantes

Altenberg [1995] demostró que el teorema de los esquemas de Holland es realmente un caso especial del teorema de Price que se usa en genética poblacional. Nótese, sin embargo, que este teorema incluye tanto términos disruptivos como constructivos.

Lee Altenberg, "**The Schema Theorem and Price's Theorem**", in L. Darrell Whitley and Michael D. Vose (Editors), *Foundations of Genetic Algorithms 3*, pp. 23–49, Morgan Kaufmann Publishers, San Mateo, California, USA, 1995.

# Otros Resultados Teóricos Importantes

También se han desarrollado versiones exactas del teorema de los esquemas [Stephens & Waelbroeck, 1999], aunque éstas siguen siendo imprácticas de usar aún para problemas de prueba relativamente simples.

Sin embargo, su uso comienza a ofrecer perspectivas nuevas muy interesantes.

Chris Stephens and Henri Waelbroeck, "**Schemata evolution and building blocks**", *Evolutionary Computation*, Vol. 7, No. 2, pp. 109–124, Summer 1999.

# Otros Resultados Teóricos Importantes

Otro de los enfoques interesantes que se han abordado en los últimos años es la identificación de bloques constructores.

El **messy GA** fue el primer intento por construir un algoritmo que explícitamente discriminara entre esquemas competitivos con base en su aptitud estimada (es decir, que operara con esquemas y no con cadenas totalmente definidas).

David E. Goldberg, Bradley Korb and Kalyanmoy Deb, "**Messy Genetic Algorithms: Motivation, Analysis, and First Results**", *Complex Systems*, Vol. 3, No. 5, pp. 493–530, 1989.

# Otros Resultados Teóricos Importantes

Munetomo y Goldberg [1999] identifican tres enfoques para la identificación de grupos de enlazamiento (**linkage**).

El primero de ellos se refiere a la "detección directa de sesgo en distribuciones probabilísticas" y su ejemplificación son los llamados *algoritmos de estimación de distribución* (DEAs, en inglés).

Masaharu Munetomo and David E. Goldberg, "**Linkage Identification by Non-monotonicity Detection for Overlapping Functions**", *Evolutionary Computation*, Vol. 7, No. 4, pp. 377–398, December 1999.

# Otros Resultados Teóricos Importantes

Los DEAs identifican una factorización del problema en un número de subgrupos de tal manera que se minimice un cierto criterio estadístico dado, con base en la población actual.

Esto corresponde a aprender un modelo de enlazamiento del problema.

Una vez que se han derivado estos modelos, se calculan probabilidades condicionales de las frecuencias de los genes dentro de los grupos de enlazamiento, y se genera una nueva población con base en ellas, reemplazando la recombinación y la mutación de los algoritmos evolutivos tradicionales.

# Otros Resultados Teóricos Importantes

Nótese que los DEAs, sin embargo, no se basan en análisis a partir de esquemas, sino en modelado estadístico.

Los otros 2 enfoques identificados por Munetomo y Goldberg [1999] usan las etapas tradicionales de recombinación y mutación, pero sesgan al operador de recombinación a fin de usar información de enlazamiento.

# Otros Resultados Teóricos Importantes

Uno de los enfoques [Kargupta, 1996] usa estadísticas de primer orden basada en perturbación (por parejas) de valores alélicos para identificar los bloques de genes enlazados que manipulan los algoritmos.

Hillol Kargupta, "**The Gene Expression Messy Genetic Algorithm**" in *Proceedings of IEEE 1996 International Conference on Evolutionary Computation*, IEEE Press, Nagoya, Japan, 20-22 May, 1996.

# Otros Resultados Teóricos Importantes

El tercer enfoque no calcula estadísticas en las interacciones de los genes con base en perturbaciones, sino que adapta los grupos de enlazamiento explícita o implícitamente vía la adaptación de operadores de recombinación [Harik & Goldberg, 1997].

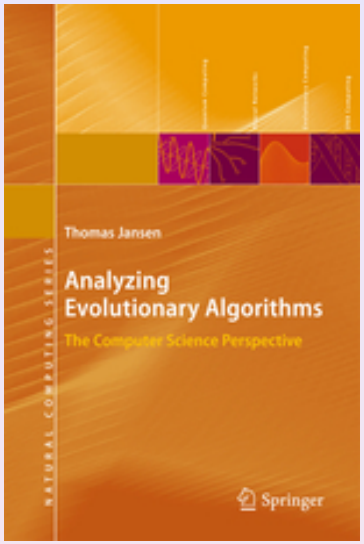
Georges Harik and David E. Goldberg, "**Learning linkage**", in Richard Belew and Michael D. Vose, (Editors), *Foundations of Genetic Algorithms 4*, pp. 247–262, Morgan Kaufmann Publishers, San Mateo, California, USA, 1997.

# Otros Resultados Teóricos Importantes

Smith [2002] presentó un modelo matemático de los enlazamientos de diferentes operadores, junto con una investigación de cómo debiera ocurrir la adaptación del enlazamiento a un nivel apropiado.

J.E. Smith, "**On appropriate adaptation levels for the learning of gene linkage**", *Genetic Programming and Evolvable Machines*, Vol. 3, No. 2, pp. 129–155, 2002.

# Otros Modelos Teóricos




El teorema de los esquemas hace predicciones en torno al cambio esperado en las frecuencias de los esquemas de una generación a la siguiente, pero no hace predicciones directamente sobre la composición de la población, la velocidad de convergencia de la población o la distribución de aptitudes de la población con respecto al tiempo.

# Otros Modelos Teóricos

Como un primer paso para tener una mejor comprensión del comportamiento de los algoritmos genéticos, y para poder hacer mejores predicciones, varios investigadores han construido modelos matemáticos "exactos" de algoritmos genéticos simples [Goldberg & Segrest, 1987; De Jong, 1994; Vose, 1999].

Estos modelos exactos capturan todos los detalles de un algoritmo genético simple usando operadores matemáticos.

David E. Goldberg and Philip Segrest, "**Finite Markov chain analysis of genetic algorithms**", in *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, Lawrence Erlbaum Associates, New Jersey, USA, 1987.

# Otros Modelos Teóricos

El principal exponente del denominado enfoque de los **sistemas dinámicos** para modelar algoritmos evolutivos en espacios de búsqueda finitos fue Michael Vose.

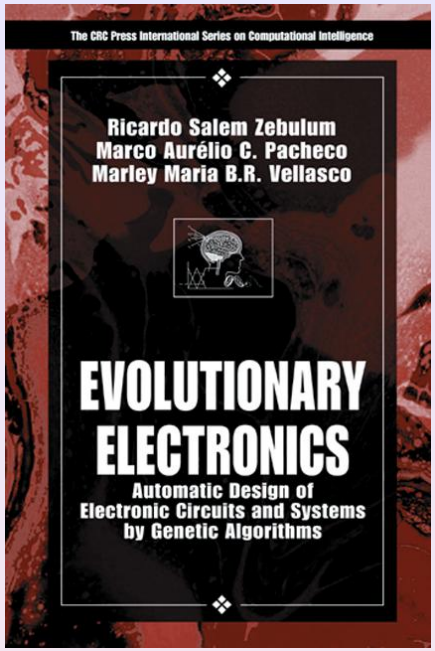
Su trabajo se enfocó fundamentalmente a algoritmos genéticos, dada su (relativa) simplicidad.

# Otros Modelos Teóricos

El enfoque de los sistemas dinámicos puede caracterizarse de la manera siguiente:

- Comenzar con un vector $n$-dimensional $\vec{p}$, donde $n$ es el tamaño del espacio de búsqueda y la componente $p_i^t$ representa la proporción de la población que es de tipo $i$ en la iteración $t$.
- Construir una **matriz de mezclado** (*mixing matrix*) $M$, que representa los efectos de la recombinación y la mutación, y una **matriz de selección** $F$, que representa los efectos del operador de selección en cada cadena para una función de aptitud dada.

# Otros Modelos Teóricos




- Componer un "operador genético" $G = F \circ M$ como el producto matricial de estas dos funciones.
- La acción del algoritmo genético para generar la siguiente población puede entonces ser caracterizada como la aplicación de este operador $G$ a la población actual: $\bar{p}^{t+1} = G\bar{p}^{t}$.

# Otros Modelos Teóricos

Bajo este esquema, la población puede verse como un punto en lo que se conoce como el "simplex" (una superficie en un espacio $n$-dimensional formado por todos los vectores posibles cuyos componentes sumen 1.0 y sean no negativos).

La forma de $G$ gobierna la manera en que una población definirá una trayectoria en esta superficie conforme evoluciona.

Una forma común de visualizar este enfoque es pensar que $G$ define un "campo de fuerza" sobre el simplex, describiendo la dirección y la intensidad de las fuerzas de evolución que actúan sobre una población.

# Otros Modelos Teóricos

La sola forma de $G$ determina qué puntos sobre la superficie actúan como **atractores** hacia los que se mueve la población, y el estudio analítico de $G$ y sus componentes $F$ y $M$ ha permitido obtener varios indicios interesantes de la forma en la que se comportan los algoritmos genéticos.

Nótese que aunque este modelo predice exactamente las proporciones *esperadas* de diferentes individuos presentes en las poblaciones evolucionadas, estos valores sólo pueden lograrse si el tamaño de la población es infinito.

De tal forma, las trayectorias que se predigan para una población inicial dada requerirán un ajuste correctivo cuando se trate de tamaños de población finitos. Esta es un área actual de investigación.

# Otros Modelos Teóricos

El análisis mediante **cadenas de Markov** es un área bien establecida en procesos estocásticos y de ahí que su uso haya llamado la atención de los teóricos de la computación evolutiva desde hace varios años.

En términos simples, decimos que un sistema puede describirse como una cadena de Markov en tiempo discreto si se cumplen las siguientes condiciones:

- El sistema puede ser caracterizado en cualquier tiempo dado por estar en uno de un número finito ($N$) de estados.
- La probabilidad de que el sistema esté en un estado dado $X^{t+1}$ en la siguiente iteración se determina únicamente por el estado en el que está en la iteración actual $X^t$, sin importar la secuencia previa de estados.

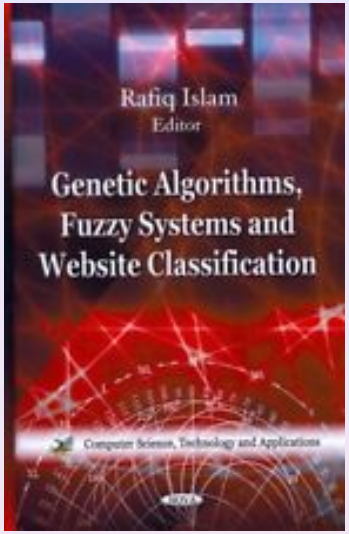
# Otros Modelos Teóricos

El impacto de la segunda condición es que podemos definir una **matriz de transición** $Q$ donde la entrada $Q_{ij}$ contiene la probabilidad de movernos del estado $i$ al estado $j$ en un solo paso ($i, j \in 1, \ldots, N$).

Es relativamente simple demostrar que la probabilidad de que después de $n$ pasos el sistema se haya movido del estado $i$ al estado $j$ está dada por la $(i, j)$-ésima entrada de la matriz $Q^n$.

Existe un buen número de teoremas y demostraciones que nos permiten hacer predicciones del comportamiento de las cadenas de Markov tales como el tiempo promedio para alcanzar un conjunto dado de estados, etc.

# Otros Modelos Teóricos



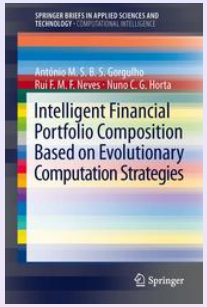

Existe un número finito de formas en las que podemos seleccionar una población de tamaño finito a partir de un espacio de búsqueda finito, de manera que podemos tratar a un algoritmo evolutivo que trabaje dentro de dicha representación como una cadena de Markov cuyos estados representan las diferentes poblaciones posibles, y diversos autores han usado este tipo de técnica para estudiar algoritmos evolutivos. De entre ellos destacan Eiben, Goldberg y Rudolph.

# Otros Modelos Teóricos

Existen propuestas para modelar convergencia de los algoritmos genéticos usando cadenas de Markov que datan de los 1980s, aunque el trabajo más conocido es el publicado por Rudolph en 1994, en el cual se demuestra que el elitismo es un operador necesario para garantizar la convergencia de un algoritmo genético simple.

Günter Rudolph, "**Convergence analysis of canonical genetic algorithms**", *IEEE Transactions on Neural Networks*, Vol. 5, No. 1, pp. 96–101, 1994.

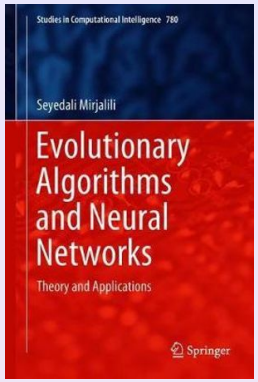
# Otros Modelos Teóricos




Otros investigadores han propuesto formulaciones exactas para las matrices de transición $Q$ de los algoritmos genéticos con representación binaria, selección proporcional, cruza de un punto y mutación uniforme.

Estas propuestas se basan esencialmente en descomponer la acción de un algoritmo genético en dos funciones: una que involucra recombinación y mutación (y la cual es puramente una función de la probabilidad de cruza y la tasa de mutación), y la otra que representa la acción del operador de selección (el cual involucra la información sobre la función de aptitud).

# Otros Modelos Teóricos




Aunque estos trabajos representan un avance significativo hacia el desarrollo de una teoría general de los algoritmos genéticos, el problema principal de este enfoque es que su utilidad se ve muy limitada.

Esto se debe a que las matrices de transición involucradas son enormes: para un problema de *l* bits, hay $\begin{pmatrix} \mu + 2^l - 1 \\ 2^l - 1 \end{pmatrix}$ poblaciones posibles de tamaño $\mu$ y la misma cantidad de filas y columnas en la matriz de transición.

# Otros Modelos Teóricos

Otro tipo de enfoques que han ganado popularidad en los últimos años son los basados en **mecánica estadística**.

La inspiración de este tipo de enfoques consiste en la observación de que los sistemas complejos realmente consisten de uniones de partes más pequeñas y que esto se ha modelado en física desde hace varios años.

En vez de intentar rastrear el comportamiento de todos los elementos de un sistema (el enfoque **microscópico**), este tipo de técnicas se enfocan a modelar el comportamiento de una cuantas variables que caracterizan el sistema. A esto se le llama el **enfoque macroscópico**.

# Otros Modelos Teóricos

Varios investigadores han usado este tipo de enfoque para modelar algoritmos evolutivos. De entre ellos destacan Adam Prügel-Bennett y Chris Stephens.

Adam Prügel-Bennett and Jonathan L. Shapiro, "**Analysis of Genetic Algorithms Using Statistical Mechanics**", *Physical Review Letters*, Vol. 72, No. 9, pp. 1305–1309, 1994.

# Otros Modelos Teóricos

Bajo este tipo de enfoque, si nos interesa la aptitud de una población que evoluciona, se derivan ecuaciones que nos proporcionen los "momentos" de la aptitud $< f >, < f^2 >, < f^3 >$ y así sucesivamente (donde los signos $<>$ denotan que la media se toma sobre el conjunto de poblaciones posibles) bajo los efectos de la selección y los operadores de variación.

A partir de estas propiedades, pueden predecirse cuestiones tales como la media, la varianza, etc. de la población evolucionada como una función del tiempo.

Nótese que estas predicciones son necesariamente aproximaciones cuya precisión depende del número de momentos modelados.

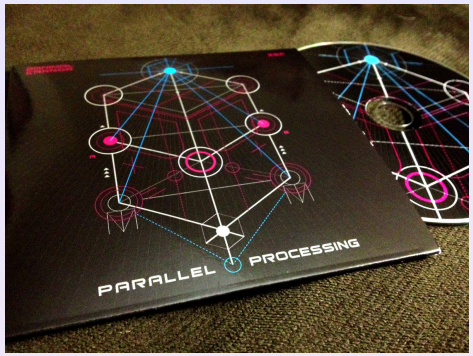
# Otros Modelos Teóricos

Las derivaciones de estas ecuaciones se basan en algunos "trucos" de la literatura de mecánica estadística y son predominantemente para una forma particular de selección (la selección de Boltzmann).

Este tipo de enfoque no pretende ofrecer predicciones que no sean la media, la varianza y demás, por lo que no se puede usar para modelar todos los aspectos del comportamiento de un algoritmo evolutivo que uno pudiese desear.

Sin embargo, estas técnicas han resultado increíblemente precisas para predecir el comportamiento de algoritmos genéticos reales en una variedad de funciones de prueba simples.

# Algoritmos Genéticos Paralelos




## Nociones de Paralelismo

Podemos definir el **procesamiento en paralelo** como la ejecución concurrente (o simultánea) de instrucciones en una computadora.

Dicho procesamiento puede ser en la forma de eventos que ocurran:

1. Durante el mismo intervalo de tiempo
2. En el mismo instante
3. En intervalos de tiempo traslapados

# Nociones de Paralelismo

La motivación más obvia del paralelismo es el incrementar la eficiencia de procesamiento.

Existen muchas aplicaciones que demandan grandes cantidades de tiempo de procesamiento y que, por ende, resultan beneficiadas de contar con arquitecturas en paralelo.

# Nociones de Paralelismo

Una de las confusiones más frecuentes respecto al paralelismo es que se cree que al contar con una computadora que tenga $n$ procesadores trabajando en el mismo problema, éste podrá resolverse $n$ veces más rápido. Esto es falso.

Al usar varios procesadores para una misma tarea, debemos tomar en cuenta que existirán:

- Problemas de comunicación entre ellos.
- Conflictos al intentar acceder a la memoria.
- Algoritmos ineficientes para implementar el paralelismo del problema.

Por tanto, aun cuando tengamos $n$ procesadores, el incremento de velocidad normalmente no será de $n$ veces.

# Nociones de Paralelismo

Existe un límite inferior respecto del incremento de velocidad real al tener $n$ procesadores.

A este límite inferior se le conoce como la **Conjetura de Minsky**, y es de $log_2 n$.

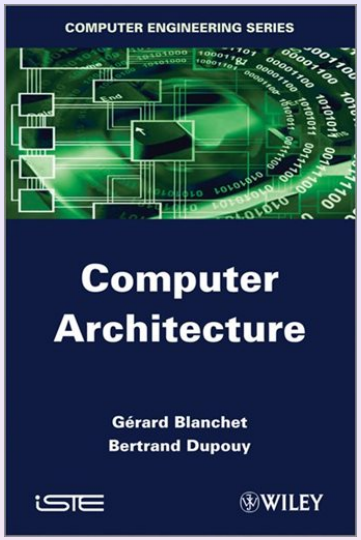
M. Minsky, "**Form and Content in Computer Science**", *Journal of the ACM*, Vol. 17, pp. 197–215, 1970.

Aunque el límite superior depende realmente de si se considera a todo el programa (incluyendo la parte de entrada y salida, la cual suele ser secuencial), suele aceptarse que éste está definido por $\frac{n}{ln\ n}$ [Hwang & Briggs, 1984].

De estos 2 límites puede inferirse fácilmente que no resulta útil agregar más y más procesadores, si lo que queremos es hacer más rápida a una computadora.

Kai Wang and Fayé A. Briggs, "**Computer Architecture and Parallel Processing**", McGraw-Hill Book Company, New York, 1984.

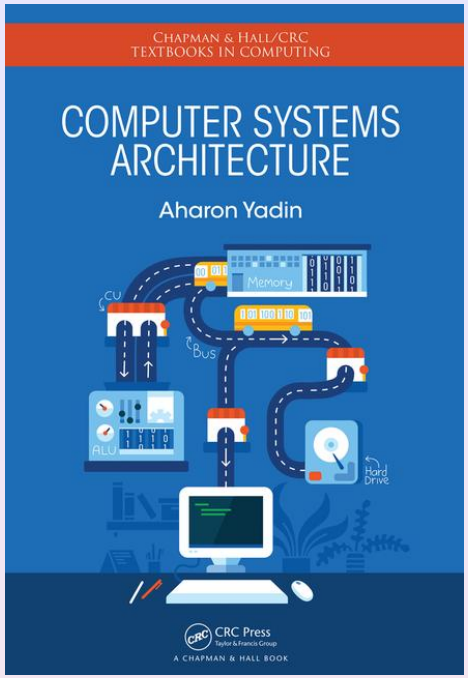
# Nociones de Paralelismo




La eficiencia de un sistema de cómputo se mide en términos de sus capacidades tanto en hardware como en software.

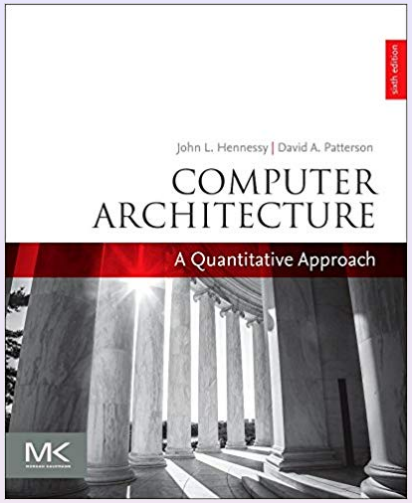
A dicha medida de eficiencia se le conoce como **rendimiento total** (*throughput*) y se define como la cantidad de procesamiento que puede realizarse en un cierto intervalo de tiempo.

# Nociones de Paralelismo




Las arquitecturas que se han utilizado para el diseño de computadoras de alta velocidad se dividen en dos grandes grupos:

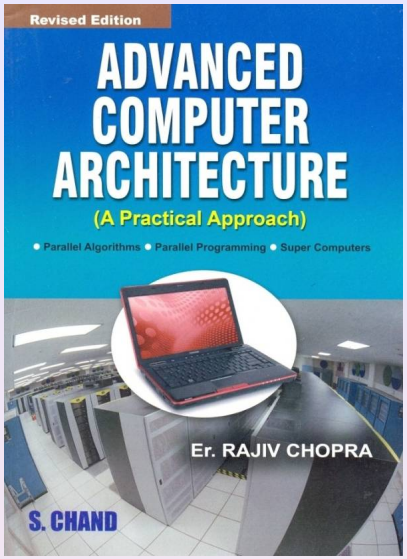
1. De propósito general
2. De propósito especial




## Computadoras de Propósito General

Las ideas de diseño más importantes que se han propuesto en torno a computadoras de alto desempeño de uso general se pueden dividir en tres grandes grupos:

1. Arquitecturas de *pipeline* (encauzamiento).
2. Multiprocesadores síncronos.
3. Computadoras de flujo de datos




## Computadoras de Propósito Especial

Dos ideas arquitectónicas representativas de los avances que se han tenido en torno a las computadoras de propósito especial son las siguientes:

1. Multiprocesadores asíncronos
2. Arreglos sistólicos

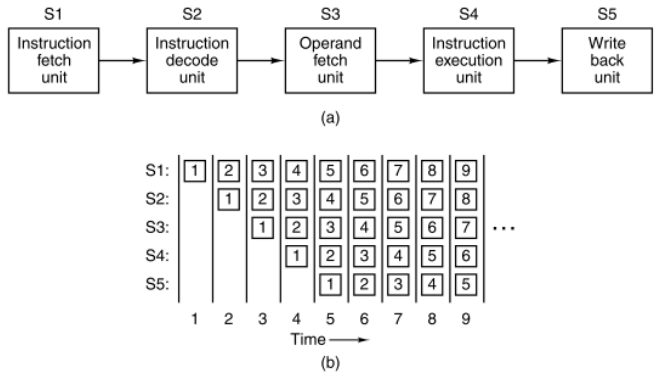


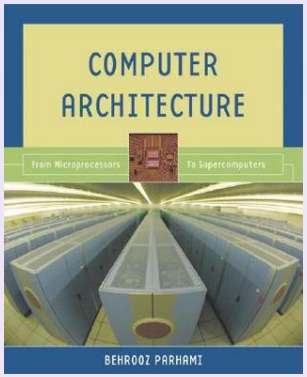

**Figure 2-4.** (a) A five-stage pipeline. (b) The state of each stage as a function of time. Nine clock cycles are illustrated.

## Arquitecturas de *pipeline* (encauzamiento)

Este proceso de encauzamiento es análogo a una línea de ensamblaje en una planta industrial.

Una función a ejecutarse por una computadora es dividida en sub-funciones más pequeñas, y se diseña hardware separado (llamado **etapa**) para cada una de estas subfunciones.

Estas etapas están conectadas entre sí, de manera que forman un solo cauce (o *pipeline*) que realiza la función original.




## Multiprocesadores Síncronos

Una máquina basada en un modelo de cómputo paralelo síncrono consiste de varios elementos de procesamiento, los cuales ejecutan la misma instrucción sobre datos diferentes.

Las instrucciones normalmente son obtenidas y transmitidas a todas las unidades de procesamiento por una unidad de control común.

Los elementos de procesamiento individuales están conectados vía una red de intereconexión para llevar a cabo la comunicación de datos entre ellos.

## Multiprocesadores Asíncronos

Se basan en el uso de varias CPUs y bancos de memoria conectados a través de un bus de datos o de una red.

En estos sistemas, cada CPU opera de manera independiente del flujo de trabajo que recibe.

El procesamiento se realiza en paralelo, utilizando las CPUs de acuerdo a su disponibilidad.

Se requiere que cada tarea se divida en sub-tareas que sean manejadas por separado por los procesadores disponibles.

## Multiprocesadores Asíncronos

Con base en los aspectos de comunicación, son posibles las siguientes arquitecturas multi-procesador básicas:

1. **Multiprocesadores con memoria compartida:** Usan un solo bus de datos, pero hay un límite en el número de procesadores que se pueden operar de manera efectiva en paralelo.
2. **Multiprocesadores basados en mensajes:** Puede usarse un número ilimitado de procesadores, pero el costo de las comunicaciones puede acabar por degradar el desempeño.

## Multiprocesadores Asíncronos

3. **Esquemas híbridos que usan tanto memoria compartida como mensajes:** Son bastante comunes en sistemas comerciales. Usan un bus común para acceder a la memoria global, al disco y los sistemas de entrada/salida, mientras que adoptan un bus separado (o una red) para manejar el tráfico de la memoria del procesador.
4. **Cómputo basado en clusters:** Son una red conectada con tecnología estándar y que se usan como computadoras paralelas basadas en mensajes.

Los clusters son el tipo de esquema de paralelismo más común. Bajo esta arquitectura, cada procesador puede ejecutar un flujo diferente de instrucciones sobre su propio flujo de datos.

## Computadoras de Flujo de Datos

Dennis [1980] sugirió un nuevo enfoque para procesamiento paralelo de grano fino, basado en el modelo de cómputo de flujo de datos.

En este modelo, se utilizan varios operadores de flujo de datos, cada uno de los cuales es capaz de realizar una operación.

Un programa para este tipo de máquina es un grafo de conexión de los operadores.

Los operadores forman los nodos del grafo, mientras que los arcos representan el movimiento de los datos entre los nodos.

J.B. Dennis, "**Data Flow Supercomputers**", *IEEE Computer*, Vol. 13, No. 11, pp. 48–56, November 1980.

## Computadoras de Flujo de Datos

Un arco es etiquetado con un token para indicar que contiene los datos. Un token se genera sobre la salida de un nodo cuando calcula la función basada en los datos de sus arcos de entrada.

A esto se le denomina “disparar” el nodo. Un nodo sólo puede disparar cuando todos sus arcos de entrada tienen tokens y no hay ningún token en el arco de salida.

Cuando un nodo dispara, remueve los tokens de entrada para indicar que los datos se han consumido.

## Computadoras de Flujo de Datos

Usualmente, el procesamiento comienza con la llegada de los datos en los nodos de entrada del grafo.

Los grafos de flujo de datos muestran las dependencias de datos en el cómputo y, por tanto, el procesamiento progresa de acuerdo a la disponibilidad de los datos.

Debido a que la secuencia de ejecución se ordena de acuerdo al flujo de datos, a las máquinas basadas en esta idea se les denomina **máquinas de flujo de datos**, para contrastarlas con las tradicionales, que se denominan de flujo de control.

## Arreglos Sistólicos

Fueron propuestos por Kung [1988]. El advenimiento de la VLSI (*Very Large Scale of Integration*) ha hecho posible el desarrollo de arquitecturas especiales, adecuadas para implementarse directamente en VLSI.

Las arquitecturas sistólicas son básicamente *pipelines* operando en una o más dimensiones.

El nombre **sistólico** se deriva de la analogía de la operación del sistema de circulación sanguínea a través del corazón.

Este tipo de arquitecturas se han usado principalmente para diseñar procesadores para gráficos, señales y procesamiento de imágenes.

S.Y. Kung, **VLSI Array Processors**, Prentice-Hall, USA, 1988.

# Nociones de Paralelismo

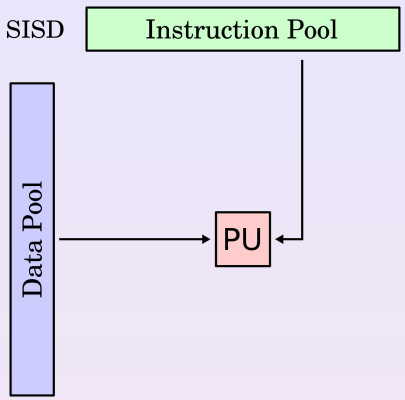
Michael J. Flynn (1966) introdujo un esquema para clasificar la arquitectura de una computadora basado en la forma en la que la máquina relaciona sus instrucciones con los datos que procesa. Flynn definió el término **stream** (*flujo*) como una secuencia de elementos, ya sea datos o instrucciones, ejecutados u operados por un procesador.

La clasificación de Flynn es la siguiente:

- **SISD**: *Single Instruction Stream, Single Data Stream*
- **SIMD**: *Single Instruction Stream, Multiple Data Stream*
- **MISD**: *Multiple Instruction Stream, Single Data Stream*
- **MIMD**: *Multiple Instruction Stream, Multiple Data Stream*

M. J. Flynn, "**Very High Speed Computer System**", *Proceedings of IEEE*, Vol. 54, pp. 1901–1909, 1966.

# Nociones de Paralelismo

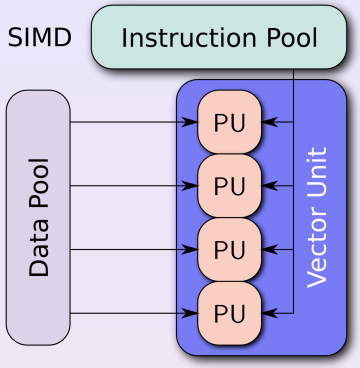



Una arquitectura **SISD** corresponde a una computadora serial convencional que todos conocemos, en la cual las instrucciones se ejecutan una por una.

En este caso, se usa una sola instrucción para lidiar con, cuando mucho, una operación sobre los datos.

Aunque es posible introducir cierto nivel de paralelismo en estas computadoras (usando *pipelining*), la naturaleza secuencial de la ejecución de sus instrucciones la coloca en esta categoría.

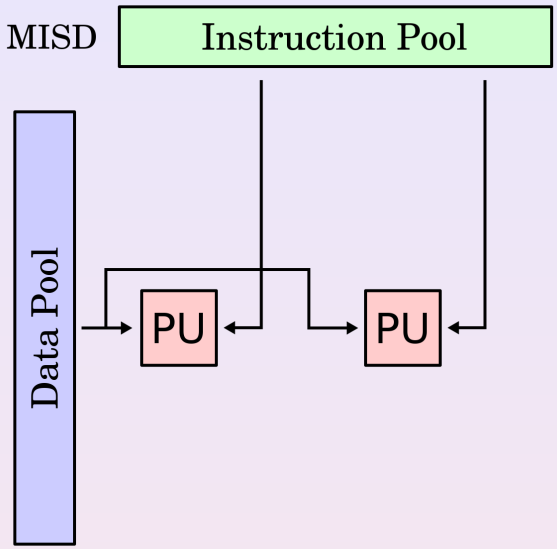
# Nociones de Paralelismo




En una computadora **SIMD**, una sola instrucción puede iniciar un gran número de operaciones.

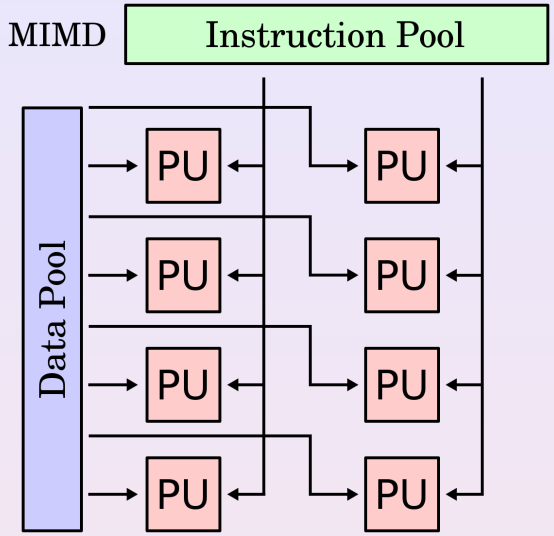
Estas instrucciones (llamadas **vectoriales**) se ejecutan de manera secuencial (una a la vez), pero son capaces de trabajar sobre varios flujos de datos a la vez.

También en este caso es posible usar *pipelining* para acelerar la velocidad de procesamiento.




La clase **MISD** implica la ejecución de varias instrucciones operando simultáneamente sobre un solo dato.

Este modelo es únicamente teórico, porque no existen computadoras que caigan dentro de esta categoría.




Una computadora **MIMD** se caracteriza por la ejecución simultánea de más de una instrucción, donde cada instrucción opera sobre varios flujos de datos.

Ejemplos de esta arquitectura son los sistemas multiprocesadores.

# Algoritmos Genéticos Paralelos

## Paralelización Global

El método más simple de paralelizar un algoritmo genético es la llamada **paralelización global**.

En este caso, sólo hay una población, como en el algoritmo genético convencional, pero la evaluación de los individuos y los operadores genéticos se paralelizan de forma **explícita**.

Puesto que sólo hay una población, la selección considera a todos los individuos y cada individuo tiene oportunidad de aparearse con cualquier otro (o sea, hay **apareamiento aleatorio**).

Por lo tanto, el comportamiento del algoritmo genético simple permanece sin cambios.