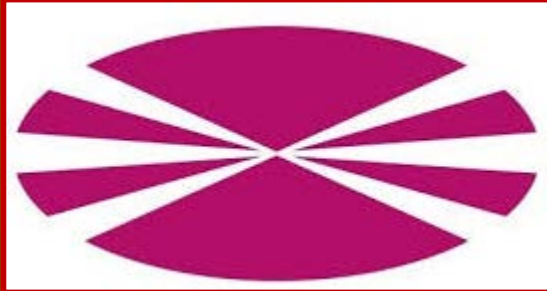
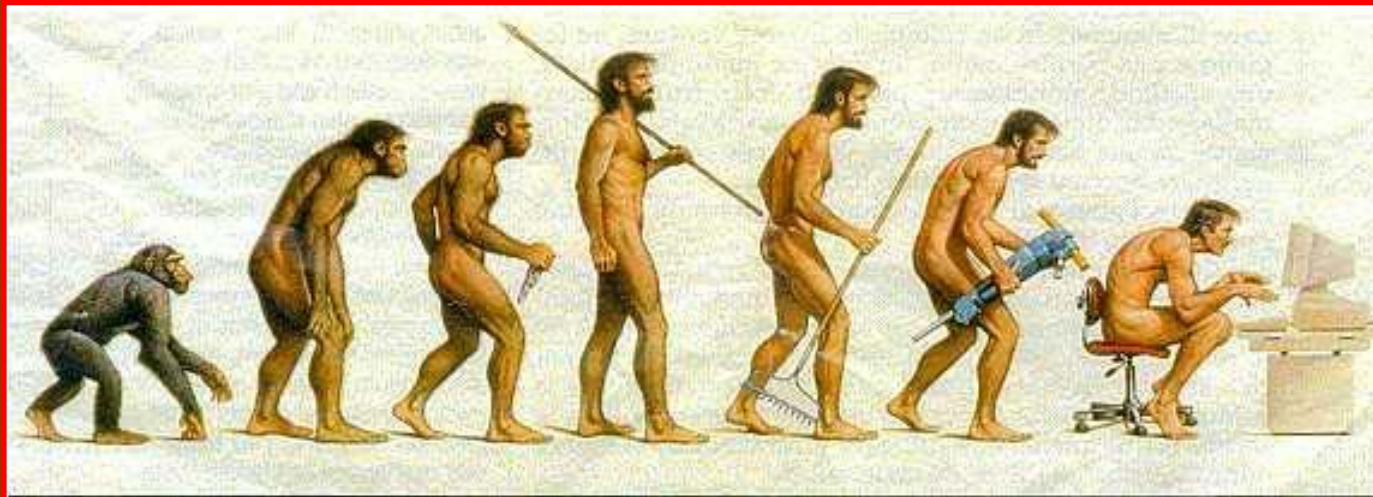


Profesores: Julián Dorado
Juan R. Rabuñal



Computación Evolutiva
Curso 2019-2020

Sistemas Inteligentes



Bibliografía

- **Libro de referencia:**

“Introducción a los Algoritmos Genéticos y la Programación Genética”. Servicio de Publicaciones de la Universidad de A Coruña. N° 140. ISBN: 978-84-9749-422-9

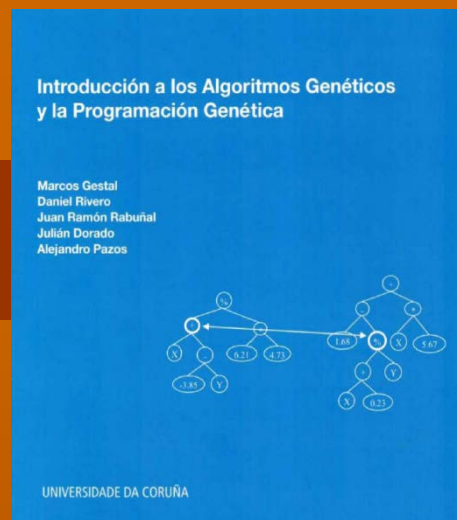
- **Introducción a la Computación Evolutiva**

—<http://nas.tic.udc.es/wiki>

- **GA y GP**

—<http://www.genetic-programming.org/>

—<http://www.geneticprogramming.com/>



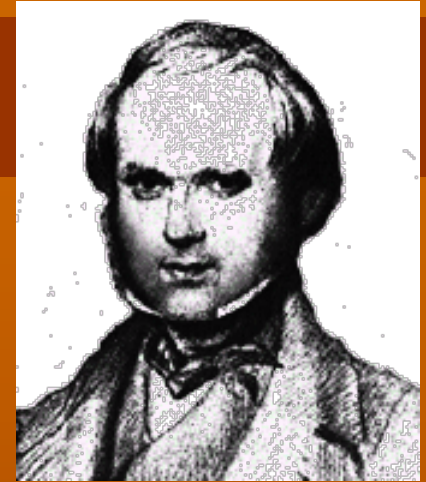
Introducción

- **Naturaleza e Informática**
 - Bioinformática
 - Computación Evolutiva
 - Genética y ecología
 - Inteligencia Artificial
- **¿Cuáles son los sistemas más complejos conocidos?**
 - Seres vivos
- **Inteligencia y Naturaleza**
 - Problema: Supervivencia
 - Respuesta: Evolución natural
- **Máquinas vs seres vivos**
 - Complejidad
 - Auto-replicación
- **¿Se puede aprovechar la forma de desarrollo de los seres vivos?**

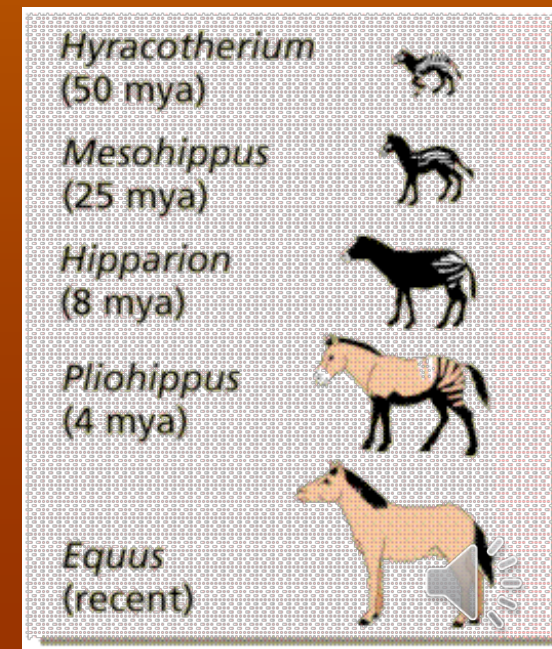


¿Qué es la evolución?

- **Proceso de optimización natural**
- **Historia:**
 - Lamarck
 - Herencia de caracteres adquiridos
 - Darwin “El origen de las especies”
 - Selección natural
 - Mendel
 - Mecanismo de herencia
 - De Vries
 - Mutación. Cambios abruptos
 - Neo-darwinismo
 - Suma de las anteriores
 - Teoría Neutralista de evolución molecular
 - Deriva genética



Charles Darwin



Ideas desde la Naturaleza

- **Aplicaciones desde la Naturaleza**

- Leonardo da Vinci

- Vuelo de pájaros y desarrollo posterior de aviones (año 1500)

- **Computación Evolutiva (CE)**

- Definir un problema

- Sin solución algorítmica

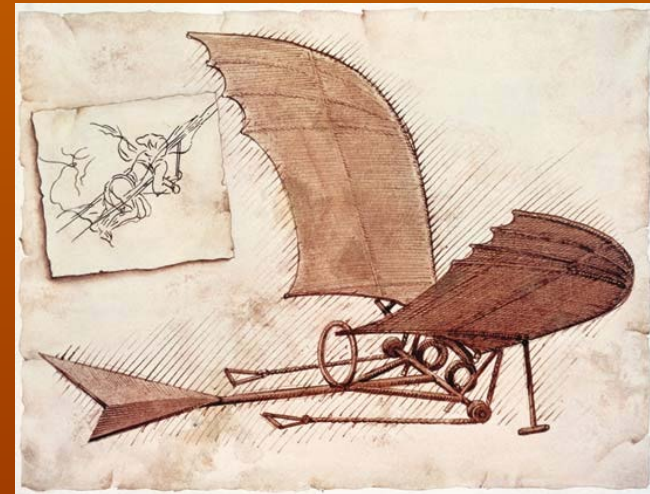
- Población de individuos o posibles soluciones

- Operadores genéticos

- Selección

- Cruce y mutación

- Sustitución





Orígenes

Basada en la teoría de la evolución de Darwin.

- John Holland, "*planes reproductivos*": técnica que incorpora la selección natural en un programa de computadora.
- Su **objetivo**: que las computadoras aprendieran por sí mismas.

Orígenes

- **John Holland** se preguntaba cómo lograba la naturaleza crear seres cada vez más perfectos.

A principios de los 60 aprendió que la evolución era una forma de adaptación más potente que el simple aprendizaje, y tomó la decisión de aplicar estas ideas para desarrollar programas bien adaptados para un fin determinado.

- **David Goldberg** conoció a Holland y se convirtió en su estudiante. Fue uno de los primeros que trató de aplicar los algoritmos genéticos a problemas industriales.

Investigadores e innovadores

- **1948** Turing: propuso la búsqueda por medio evolutivos o genéticos
- **1962** Bremermann: propone la optimización a través de la recombinación y la evolución
- **1964** Rechenberg: introdujo los conceptos de estrategias de evolución
- **1965** L. Fogel, Owens and Walsh: introdujo la programación evolutiva
- **1975** Holland: Realmente es el creador de la teoría en 1975 y re edita su libro en 1992
- **1989** D. Goldberg: Promueve el uso de algoritmos genéticos para aprendizaje y optimización
- **1992** Koza: introduce la programación genética

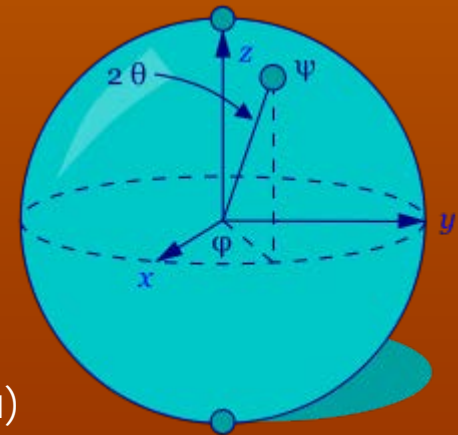
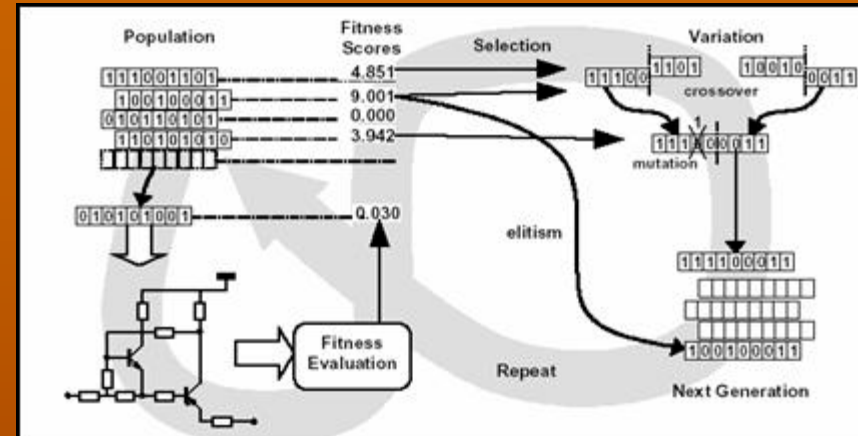
Computación Evolutiva



Técnicas en CE

- Algoritmos Genéticos
- Programación Genética
- Vida Artificial
- Otras técnicas

- Estrategias evolutivas
- Hardware Evolutivo
- Learning classifier systems
- Ant colony optimization
- Coevolución
- Artificial Immune Systems
- Algoritmos culturales (fenómenos ideológicos)
- Computación ADN (simular computador como ADN)
- Optimización Enjambre Partículas (Swarm) (abejas)



Orígenes: ¿Qué es un Algoritmo Genético?

- **¿Podemos crear un algoritmo con la misma filosofía que emplea la naturaleza?**

nacieron los algoritmos genéticos



Los algoritmos genéticos establecen una analogía entre el conjunto de posibles soluciones de un problema y el conjunto de individuos de una población natural



Orígenes: Fundamentos de los AG

- La naturaleza utiliza potentes medios para impulsar la evolución satisfactoria de los organismos, el proceso de selección natural
- Los organismos que son poco aptos para un determinado ambiente mueren, en tanto que los que están bien adaptados para vivir, **se reproducen**, y transmiten sus caracteres
- Ocasionalmente se producen **mutaciones** al azar, y aunque implican la pronta muerte del individuo mutado, algunas mutaciones dan como resultado nuevas y satisfactorias especies



Algoritmos Genéticos

Los Algoritmos Genéticos (AG) son algoritmos de búsqueda inspirados en procesos de selección natural.

Se aplican principalmente en problemas de optimización. Se comportan de un modo muy eficaz en problemas de superficie compleja, con múltiples mínimos locales y grandes espacios de búsqueda

Está justificado su uso en aquellos problemas cuya complejidad no permita una solución directa. Por ejemplo los no resolubles polinomialmente (NP-duros)

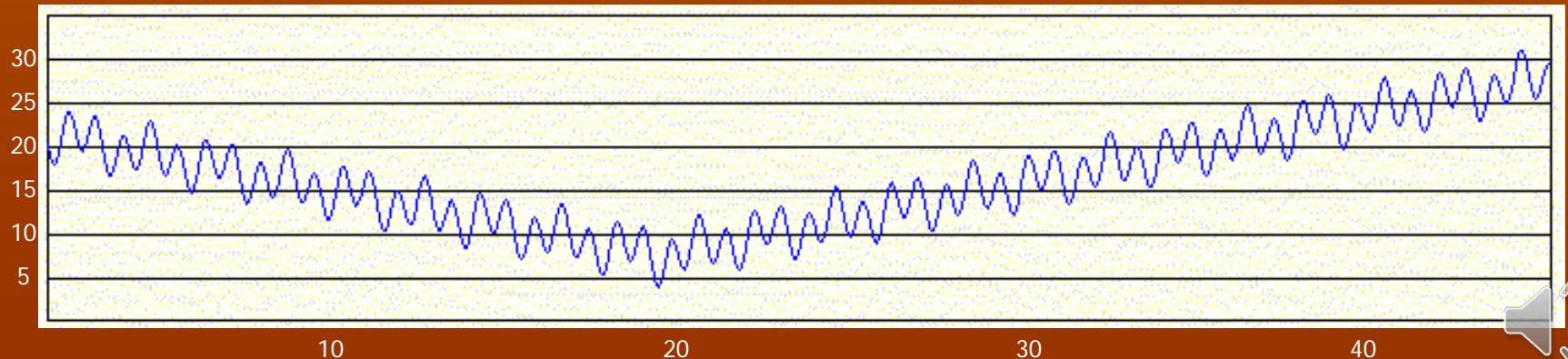
Componentes de un AG

Generalmente se acepta que un Algoritmo Genético debe tener 5 componentes:

1. Una representación genética de las soluciones del problema, es decir, representación de las variables que intervienen en cadenas de bits/caracteres
2. Una forma de crear una población inicial de soluciones
3. Una función de evaluación en términos de conveniencia o adaptación de la solución evaluada
4. Operadores genéticos que cambien la composición de los descendientes
5. Valores para los parámetros utilizados por los Algoritmos Genéticos (N , P_c , P_m , ...)

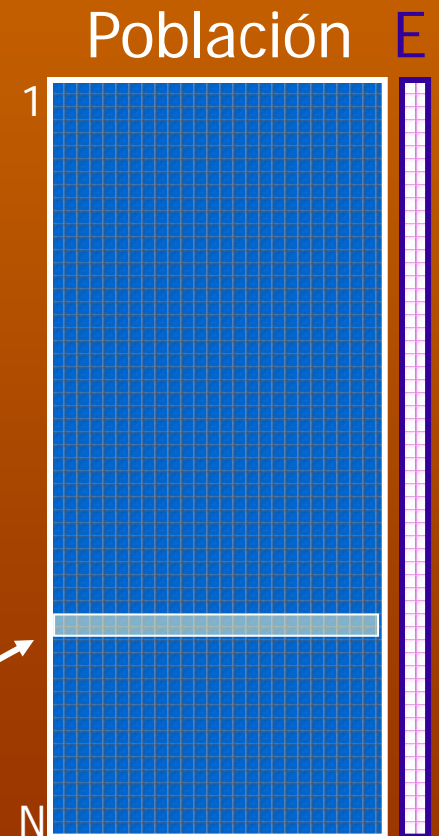
Codificación del problema

- **Individuo o solución: Cadena de bits/valores**
- **Ejemplo:**
 - Buscar el mínimo de una función definida $f(x)$
 - Codificar el valor del eje x en binario (5 bits)

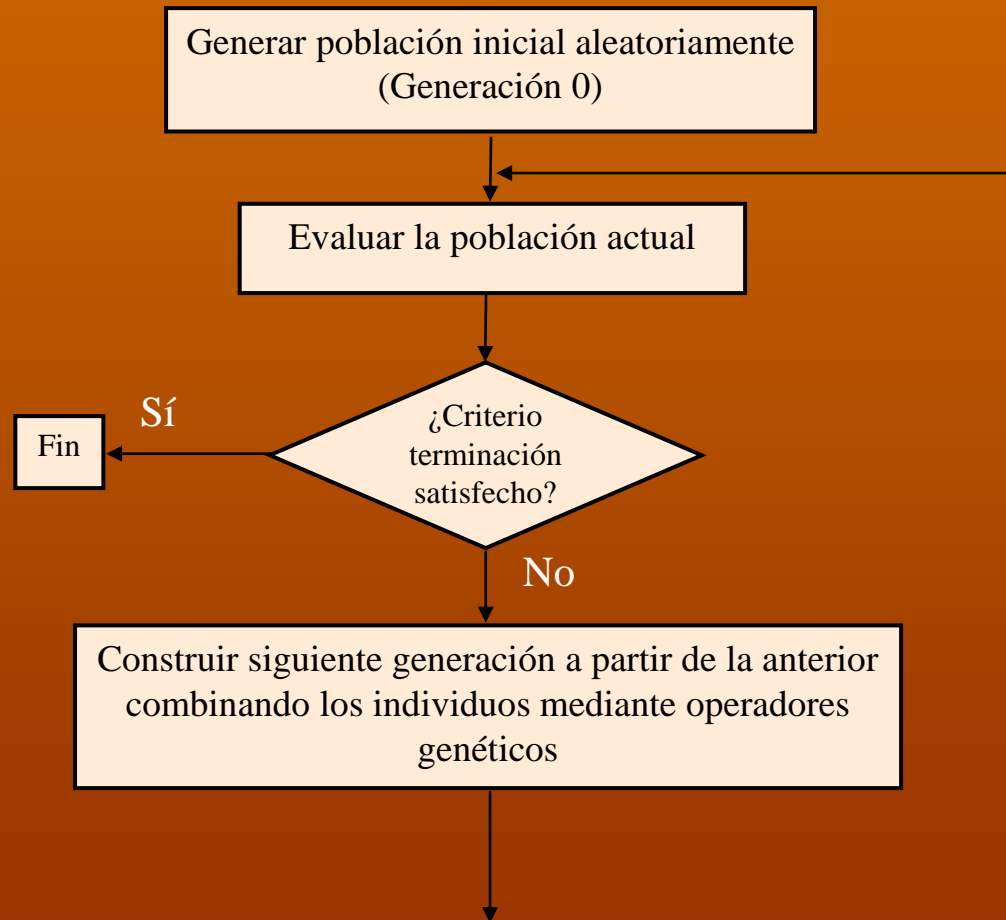


Población y Evaluación de individuos

- Se genera aleatoriamente la población inicial.
- A cada uno de los individuos o cromosomas generados se le aplicará la **función de aptitud o evaluación**.
- Esta función que debe ser capaz de "castigar" a las malas soluciones, y de "premiar" a las buenas, de forma que sean estas últimas las que se propaguen con mayor rapidez.
- Base para determinar qué soluciones tienen mayor o menor probabilidad de sobrevivir.



ESQUEMA AG



Ciclo de Generaciones

- Cada ciclo de funcionamiento se denomina **Generación**
- En cada generación:
 - Selección
 - Cruce
 - Sustitución
 - Mutación
- Se termina al alcanzar un nivel de error o un número de generaciones
 - [Ejemplo de funcionamiento](#)

Criterio de Selección

- Se procede a la selección de los que se cruzarán en la siguiente generación
- Seleccionar individuos de la población actual, generar una nueva población y reemplazar con ella a la anterior.
- También a veces se mantienen los N mejores individuos de una población a la siguiente.
- Varias formas:
 - *Ruleta*
 - *Selección de torneo*
 - *Basado en el rango*



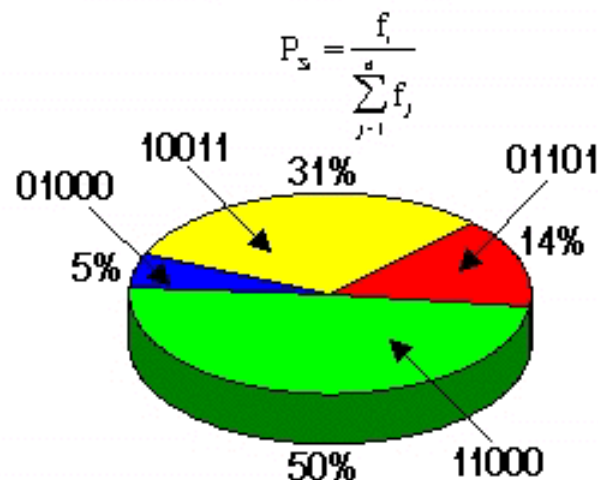
Criterio de Selección: Ruleta

Operators Example

The Problem is to Maximize $f(x) = x^2$

Number	String	Fitness	% of the Total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
Total		1170	100.0

1.- Roulette Wheel Selection



2.- One-Point Crossover (SPX)

$$P_c \in [0.6 \dots 1.0]$$

parents	offspring
01 101 (169)	01000 (64)
11 000 (576)	11101 (841)

3.- Mutation

$$P_m \in [0.001 \dots 0.1]$$

Mutate the first allele

0	1	0	0	0	1	1	0	0	0
4	3	2	1	0	4	3	2	1	0

fitness=64 \Rightarrow fitness=576



• • • Criterio de Selección: Torneo

- **Se seleccionan al azar un número fijo de individuos (2,4,8...)**
- **Se realiza un torneo entre ellos**
 - Significa que se escoge al más adaptado
- **Ventaja**
 - Evita que siempre se cuente más con los mejores individuos de la población

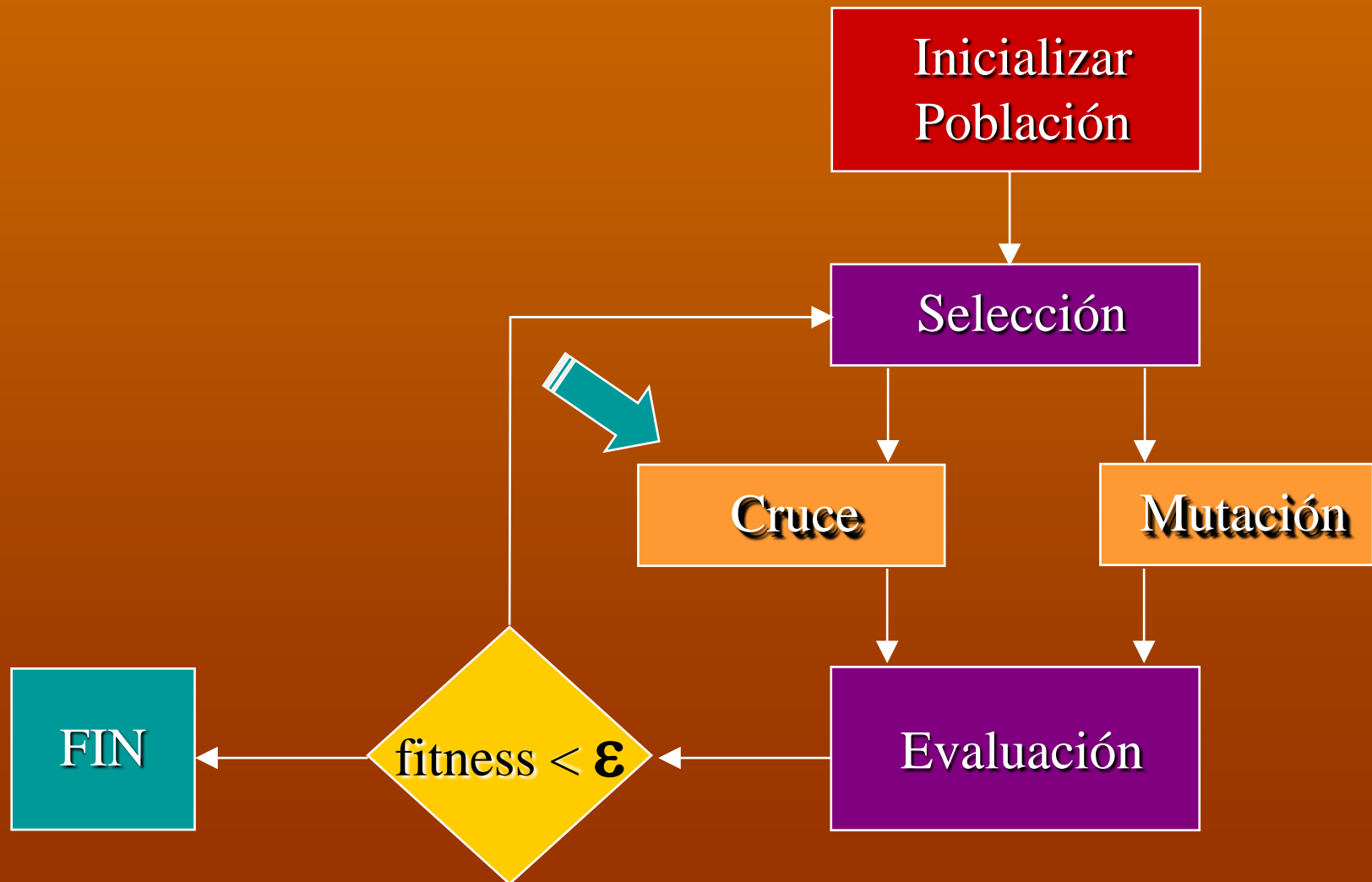


Criterios de Selección

Tipo de operador	Algoritmo	Comentario
Selección de individuos	Torneo	Reparte la búsqueda más en el espacio de estados (exploración)
	Sobranter estocástico	Realizan la búsqueda de forma intermedia entre ambos
	Universal estocástica	
	Muestreo determinístico	
	Ruleta	Pone más presión de búsqueda sobre el mejor individuo (explotación)



Estructura del Algoritmo Genético



El cruce

- Se denomina *técnica de cruce* a la forma de calcular el genoma del nuevo individuo en función del genoma del padre y de la madre
- Es responsable de las propiedades del algoritmo genético, y determinará la evolución de la población
- Técnicas básicas de cruce:
 - *Cruce básico de un punto*
 - *Cruce multipunto*
 - *Cruce uniforme*
 - *Cruce segmentado*
 - *Cruces para permutación*
 - *Cruce de mapeamiento parcial*
 - *Cruce de orden*
 - *Cruce de ciclo*



Operadores: Cruce

- Mezclar el material genético
- Objetivo: Reunir en un individuo lo mejor
- Dos individuos progenitores



Operadores: Cruce

- Mezclar el material genético
- Objetivo: Reunir en un individuo lo mejor
- Dos individuos progenitores para generar dos individuos hijos



Operadores: Cruce

- **Tipos:**

- Cruce de dos puntos

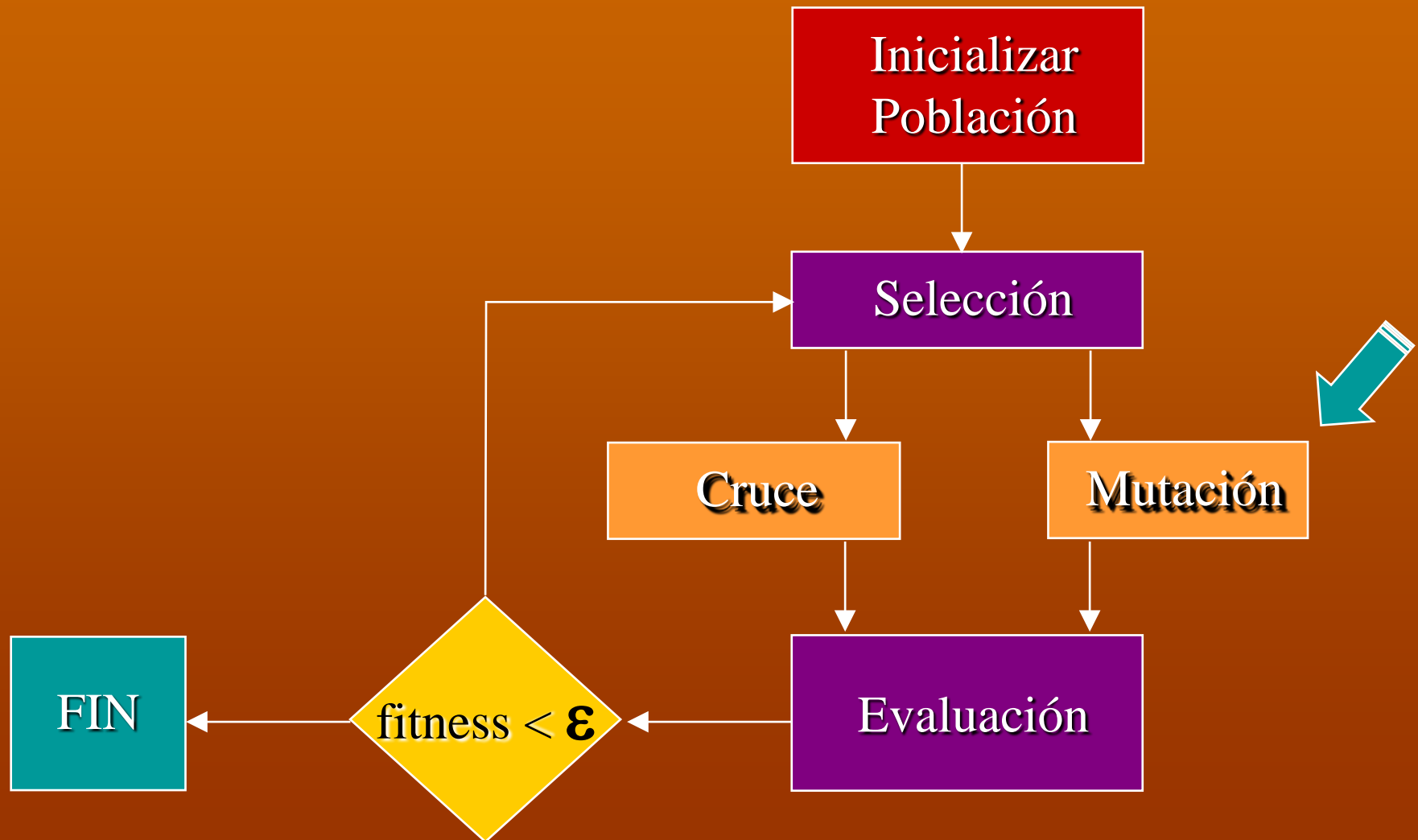


- Cruce uniforme

- Probabilidad 0.5 en cada gen de ser de cada padre



Estructura del Algoritmo Genético



Mutación

- Consiste en el cambio de un bit elegido aleatoriamente
- Opera sobre un solo individuo, determina una posición y la invierte con cierta probabilidad
- Permite salir de máximos/mínimos locales
- Contribuye a la diversidad genética de la población
- Se establece la frecuencia de mutación



Mutación

- **Se genera un número aleatorio en cada operación de mutación**
 - Si el número está por debajo de una probabilidad, se cambiará el bit, sino no
- **Existen varias técnicas distintas de mutación:**
 - *Mutación de bit*
 - *Mutación multibit*
 - *Mutación de gen*
 - *Mutación multigen*
 - *Mutación de intercambio*
 - *Mutación de barajado*



Operadores: Mutación

- **Cruce busca combinaciones**
- **Mutación busca variabilidad**

Individuo
Original

1	0	0	1	1
---	---	---	---	---



Operadores: Mutación

- **Cruce busca combinaciones**
- **Mutación busca variabilidad**

Individuo
Original



Individuo
Mutado



Cruce y mutación

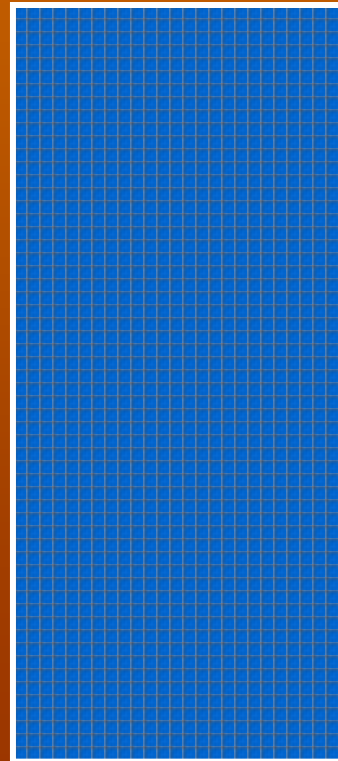
- **Balancear la búsqueda de la solución a un problema**
 - Exploración
 - Explotación
- **Buscar un compromiso entre las dos aproximaciones**
- **Ejemplo**
 - [Busqueda de máximos de una función](#)
 - Ajedrez
 - <http://www.vanheusden.com/pos/>



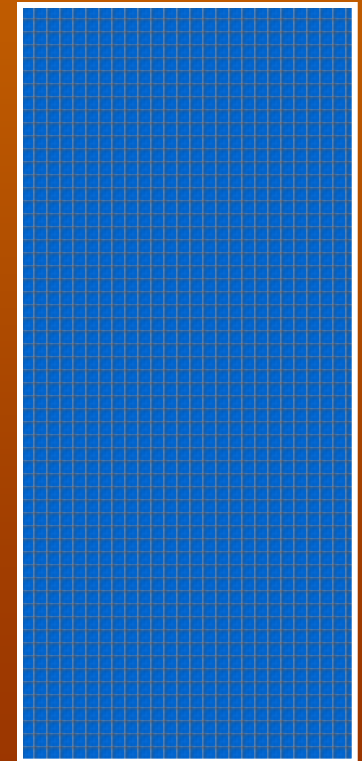
Sustitución

- **Objetivo:**
Mantener el tamaño de la población
- **Tipos de gestión de población**
 - Generacional:
genera una nueva población
 - Selección por ruleta
 - Aplicar probabilidad de cruce

Población
Inicial



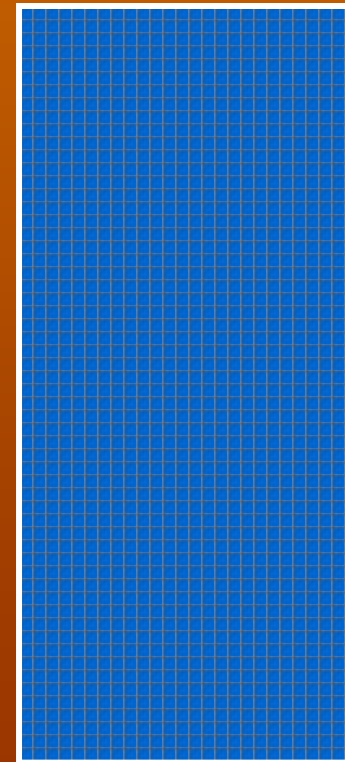
Población
Final



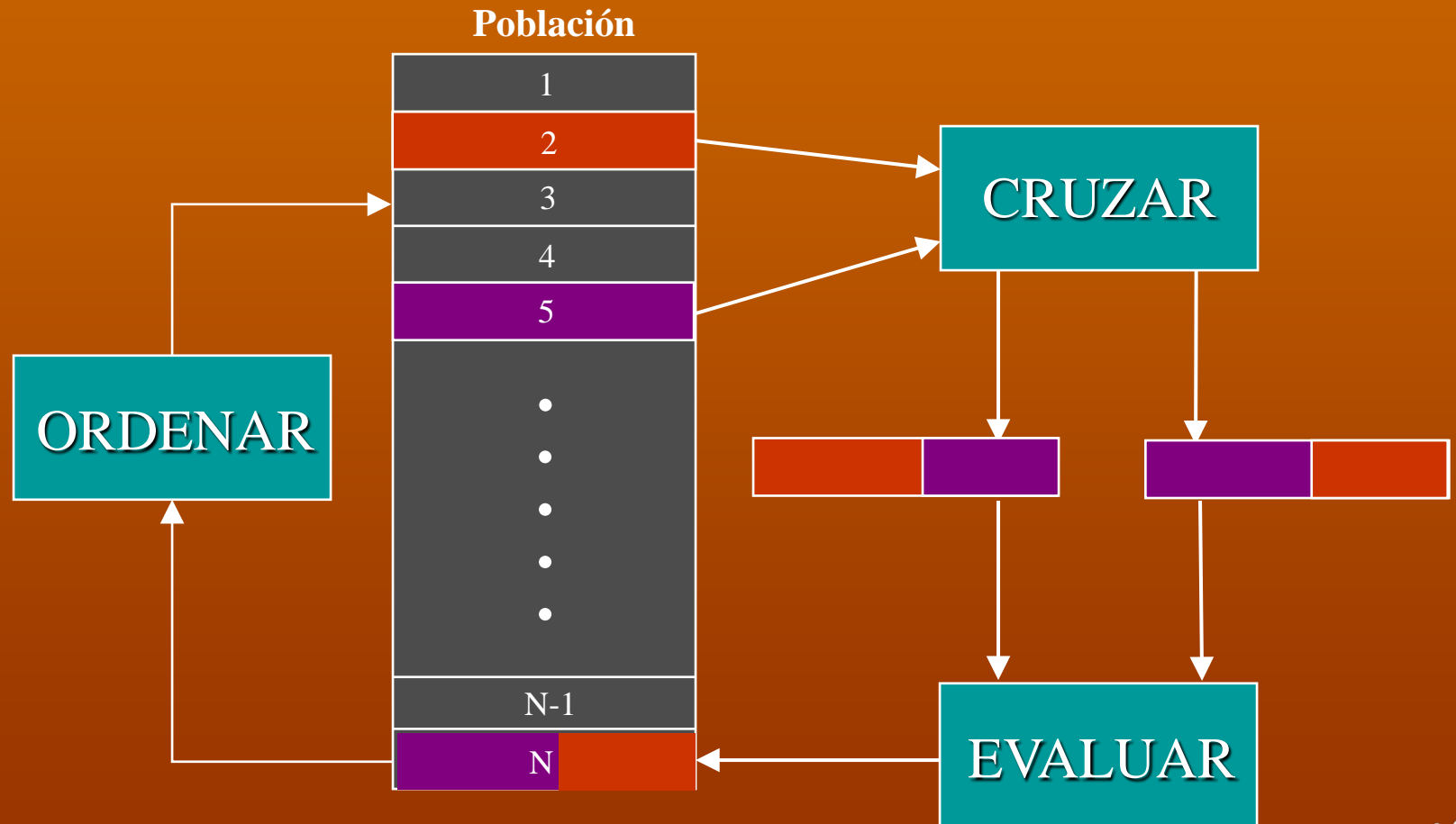
Sustitución

- **Objetivo: Mantener el tamaño de la población**
- **Tipos de gestión de población**
 - Steady-state: sustituye individuos
 - Tipos de Sustitución de individuos
 - Peores: Provoca la homogenización de la población
 - Padres
 - Parecidos
 - Elitismo: mantener los mejores individuos

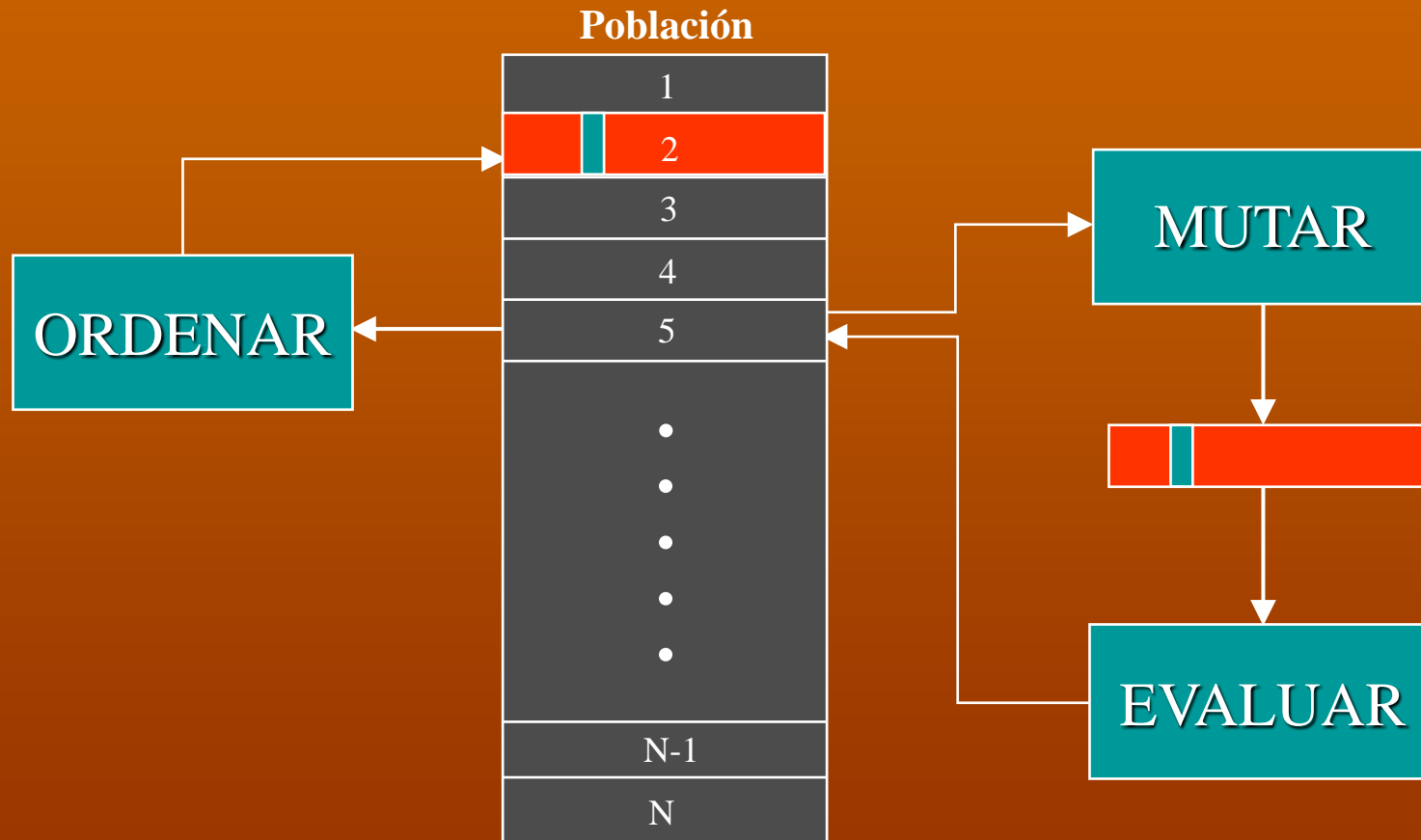
Población



AG Steady-state (Cruce) con sustitución de peores

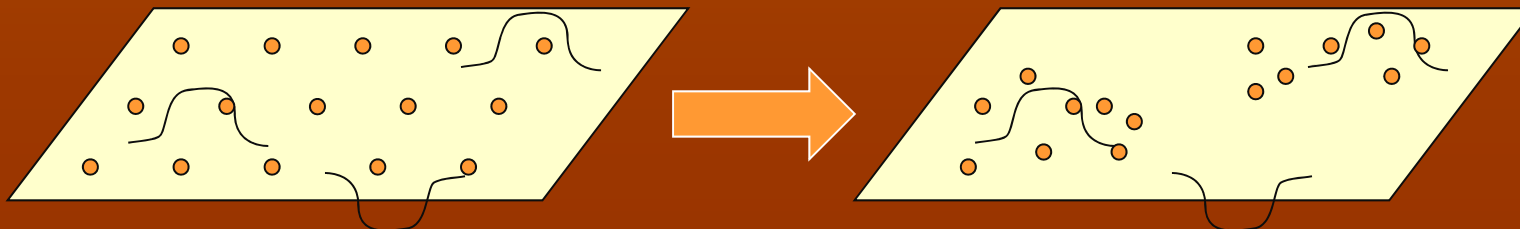


AG Steady-state (Mutación)



Función de evaluación

- **Las técnicas de CE no garantizan la consecución del óptimo global**
 - Distribución inicial uniforme sobre el espacio de soluciones
 - Consiguen una convergencia muy rápida a la zona del óptimo global
- **Combinar con técnicas de optimización local**
 - Escalado de colinas (Hill climbing)



•
•
•

Criterio de paro

- **Cuando un porcentaje alto de la población converge a un valor.**
- **Si con ese valor no se llega a la medida esperada, entonces:**
 - se toma una pequeña proporción y se inyecta “diversidad genética”
 - o se reemplaza completamente la población.

•
•
•

Criterios sobre los parámetros

- **Tamaño de población**
 - Balance entre una población muy pequeña y una población muy grande
 - Normalmente se elige una población de tamaño fijo. Cientos de individuos
 - También existen esquemas de poblaciones de tamaño variable
- **Porcentaje de cruce y mutación**
 - Cruce 90% mutación 5-10%
 - Valores variables según fases del algoritmo
 - Al inicio y al final de la simulación interesa
 - cruce ¿alto/bajo? y mutación ¿alto/bajo?

-
-
-

RESUMEN: PASOS PARA APLICAR UN AG

- Codificación
 - Cómo se representa una solución para que pueda ser usada por un AG
- Función de ajuste
 - Cómo se evalúa la bondad de una solución
- Configuración de parámetros:
 - Tamaño de la población
 - Tasa de cruces
 - Probabilidad de mutación
 - Algoritmo de selección
 - Algoritmo de cruce
 - etc.

•
•
•

EJEMPLOS

- En problemas de optimización
 - Funciones
 - Investigación operativa
 - Transporte: Volumen, tamaño, coste y rapidez
 - Sistemas clasificadores
 - Descubrir características para determinar
 - Selección de variables para clasificación
 - Rutas
 - Problema del viajante

-
-
-

Optimización de funciones: De Jong

De Jong construyó un ambiente de prueba de cinco problemas de minimización de funciones. Incluyo funciones con las siguientes características:

- Continuas/discontinuas
- Convexas/no convexas
- Unimodales/multimodales
- Cuadráticas/no cuadráticas
- Baja dimensionalidad/alta dimensionalidad
- Determinística/estocástica

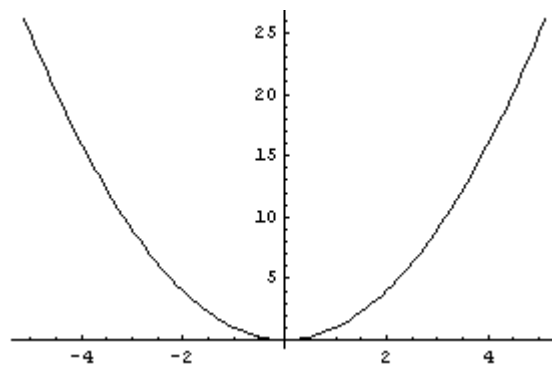
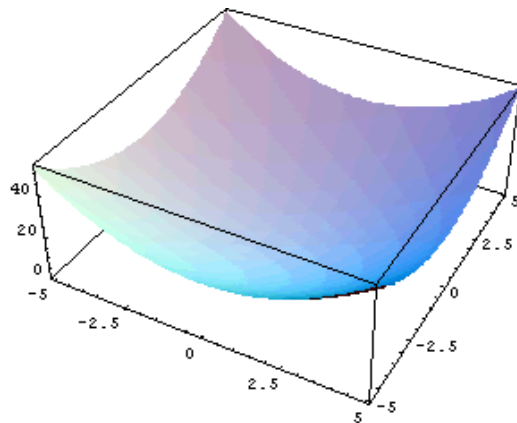
De Jong, K. (1975)

An analysis of the behaviour of a class of genetic adaptive systems

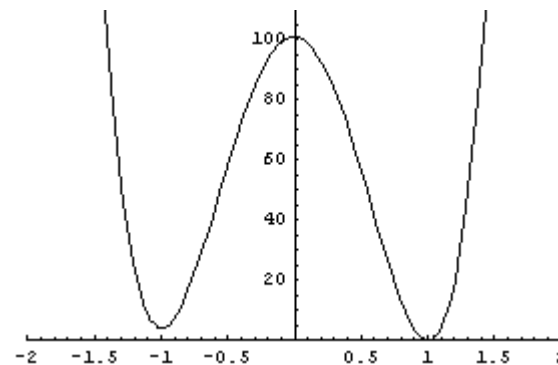
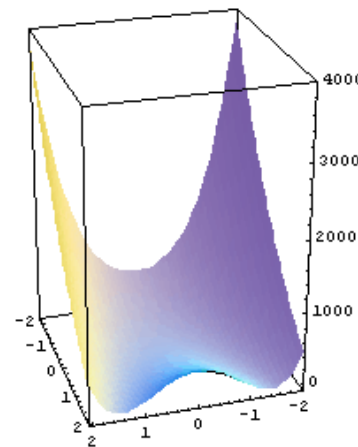
PhD thesis, University of Michigan

Funciones: De Jong

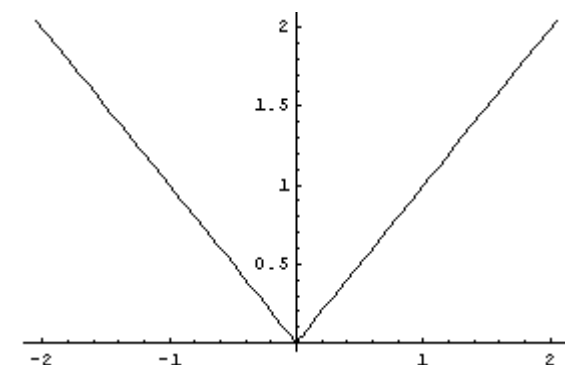
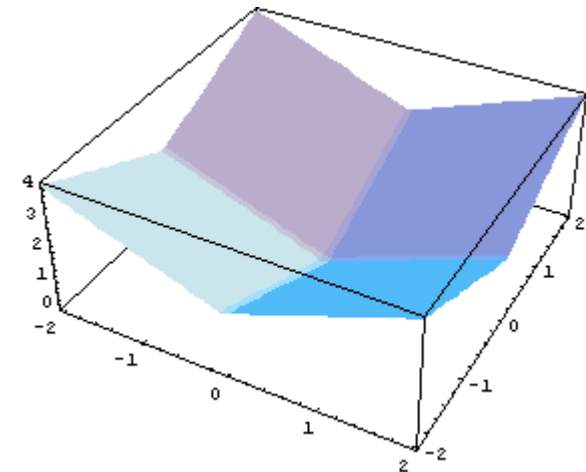
$$\sum_{l=1}^{Dba} x_l^2$$



$$\sum_{l=1}^{Dba-1} (100 (x_l^2 - x_{l+1})^2 + (1 - x_l)^2)$$

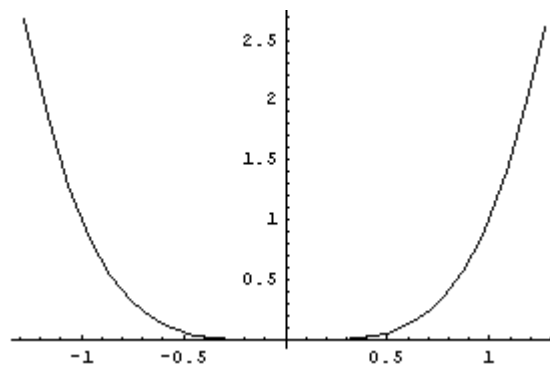
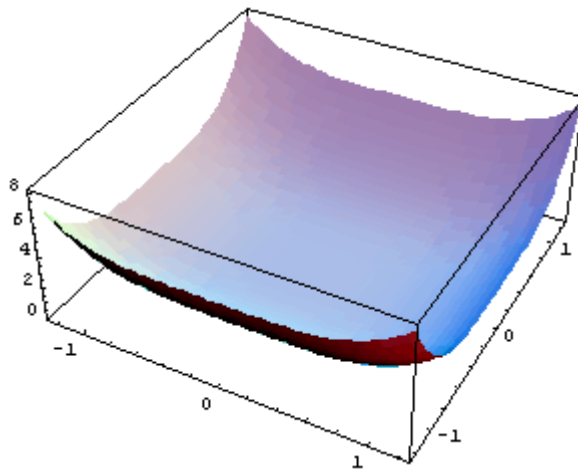


$$\sum_{l=1}^{Dba} |x_l|$$

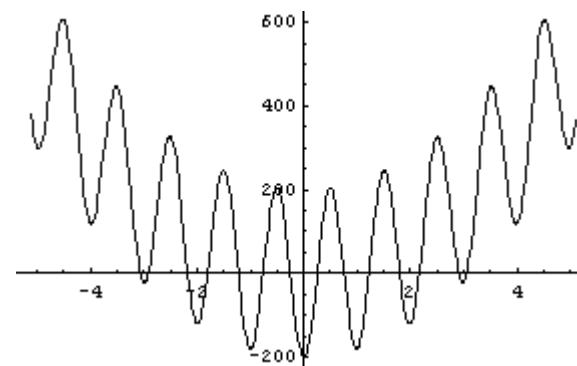
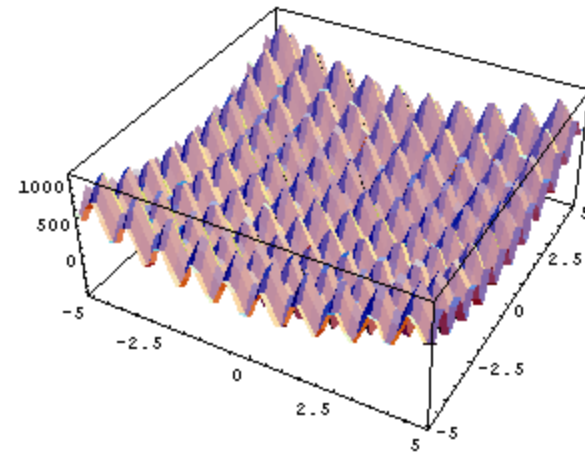


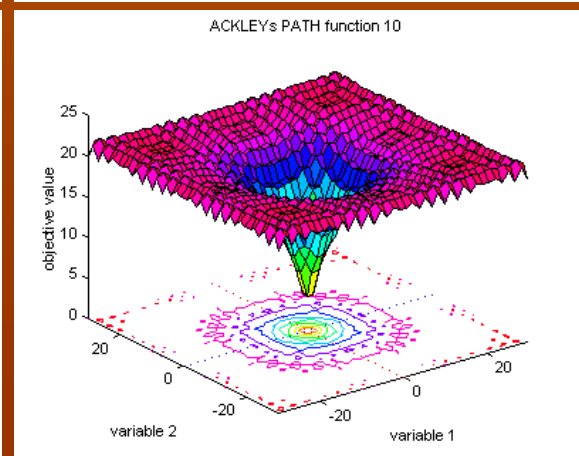
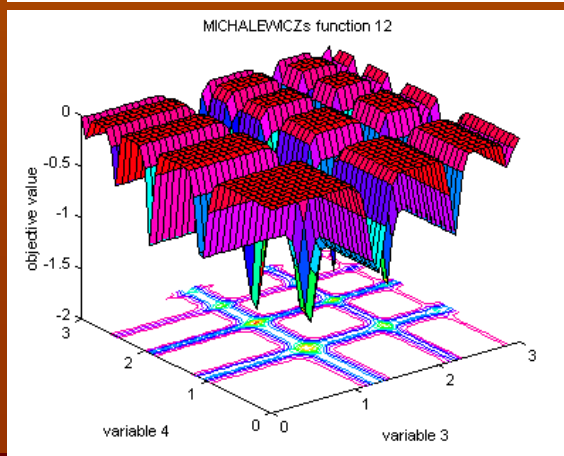
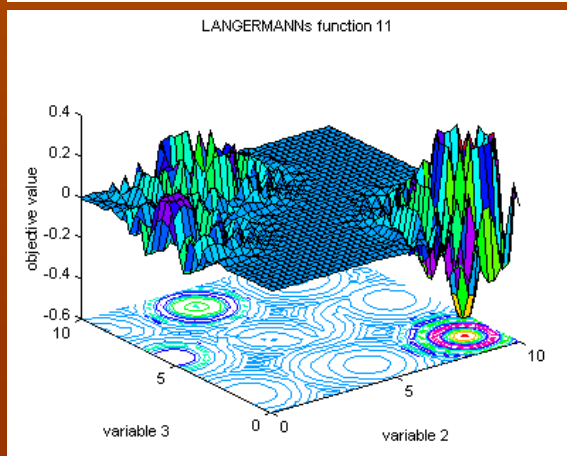
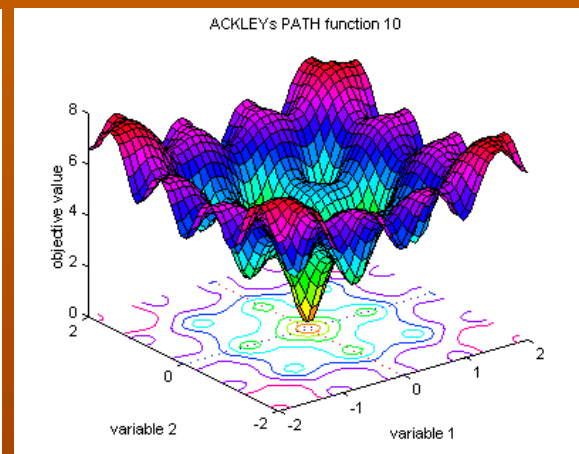
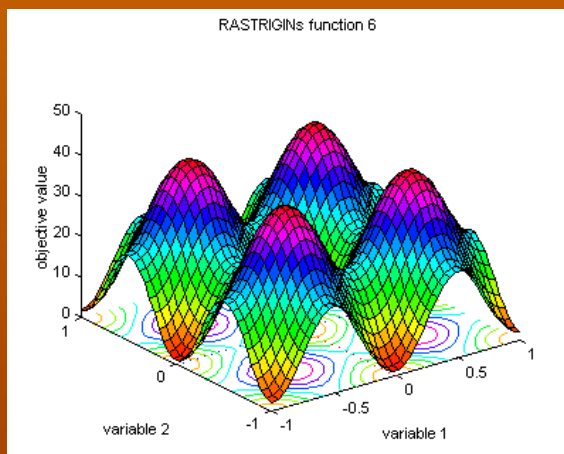
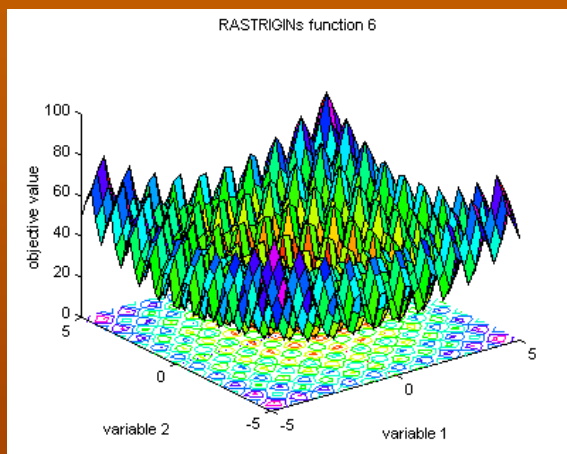
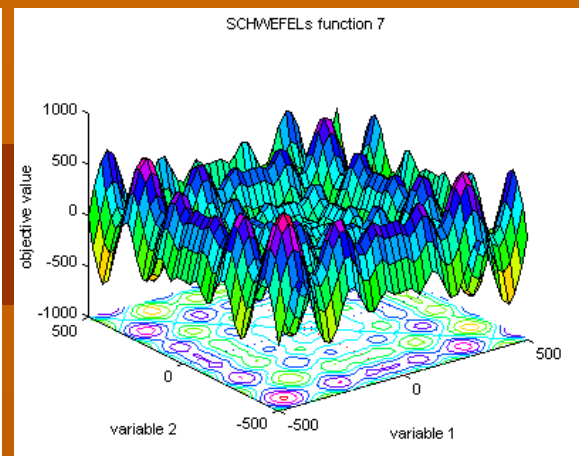
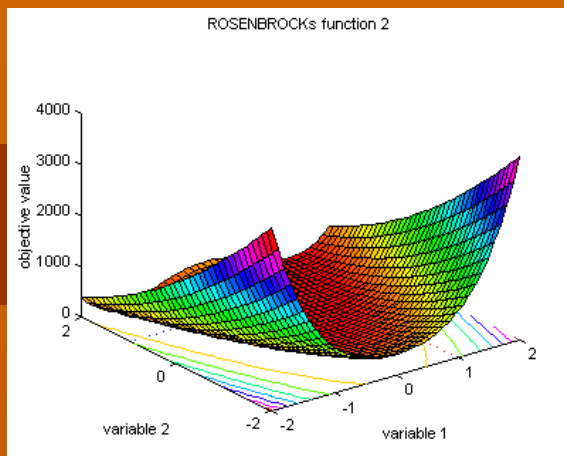
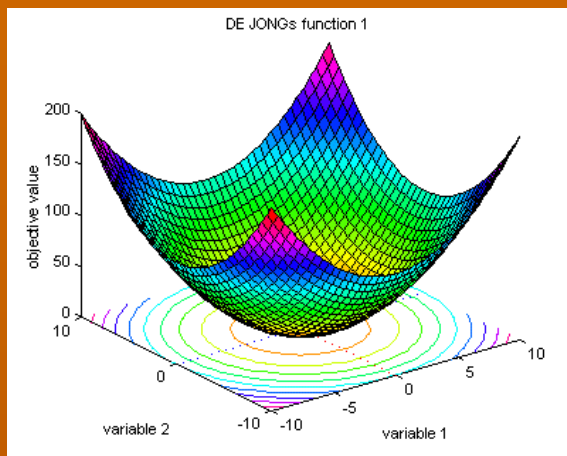
Funciones: De Jong

$$\sum_{i=1}^{Dba} i x_i^4$$



$$(20) \sum_{i=1}^{Dba} (x_i^2 - 10 \cos(2\pi x_i))$$

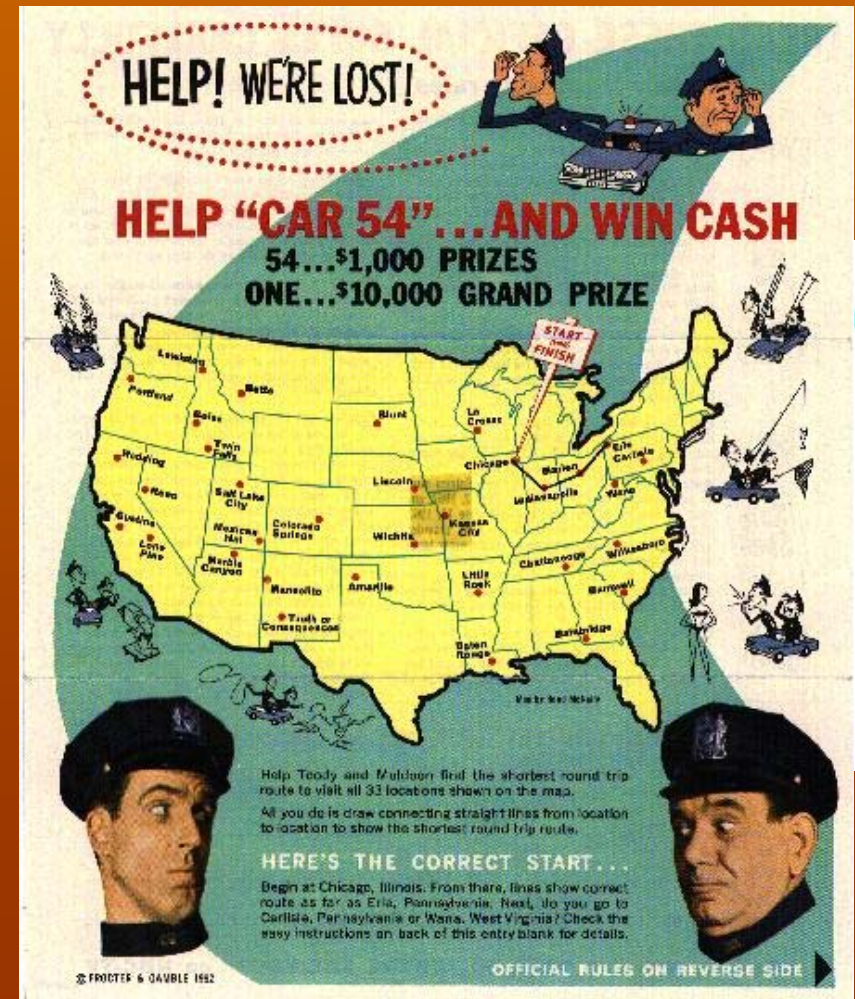




Problema del viajante (TSP)

- Traveling Salesman Problem
- Buscar la ruta óptima entre un número de ciudades
 - más corta o menos tiempo
 - sin repetir el paso
- 1962 concurso con 33 ciudades de EEUU

www.tsp.gatech.edu



Problema de optimización: El viajante

• PROBLEMA DEL VIAJANTE

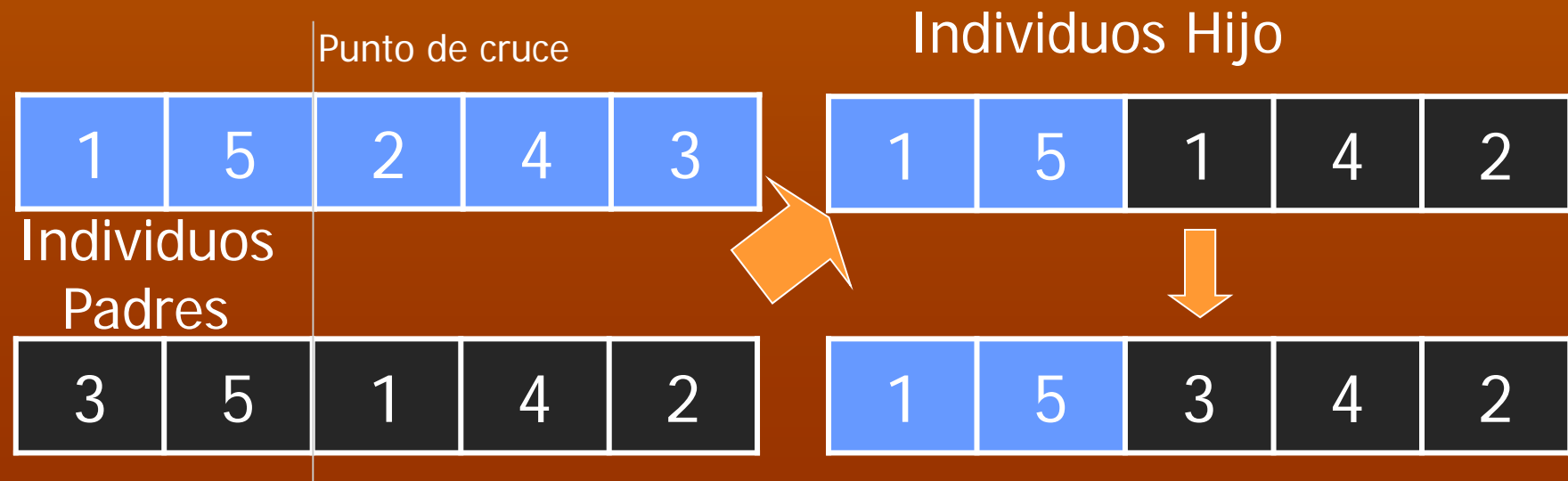
- Pasos básicos para resolverlo con AGs:
 - 1. Crear una población inicial: grupo de recorridos aleatoriamente.
 - 2. Encontrar los dos mejores en la población y combinarlos.
 - 3. Modificar operador de cruce y mutación
 - Permutaciones de ciudades
 - Cruce: escoger parte de un padre y el orden de las ciudades del otro
 - Mutación: cambiar de orden
- A veces el algoritmo llega a un punto donde todos los individuos son iguales. Para que no ocurra esto:
 - Partir de una población inicial grande y
 - Utilizar el método de mutación
- Ejemplo: Se consideran 5 ciudades:



•
•
•

Operadores: Cruce

- Mezclar el material genético
- Objetivo: Reunir en un individuo lo mejor
- Dos individuos progenitores

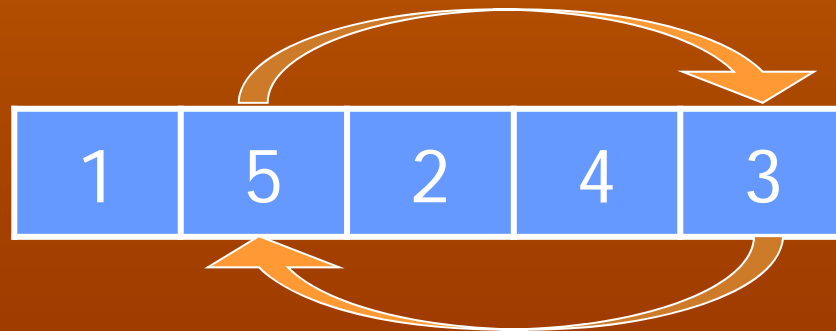


•
•
•

Operadores: Mutación

- **Cruce busca combinaciones**
- **Mutación busca variabilidad**

Individuo
Original

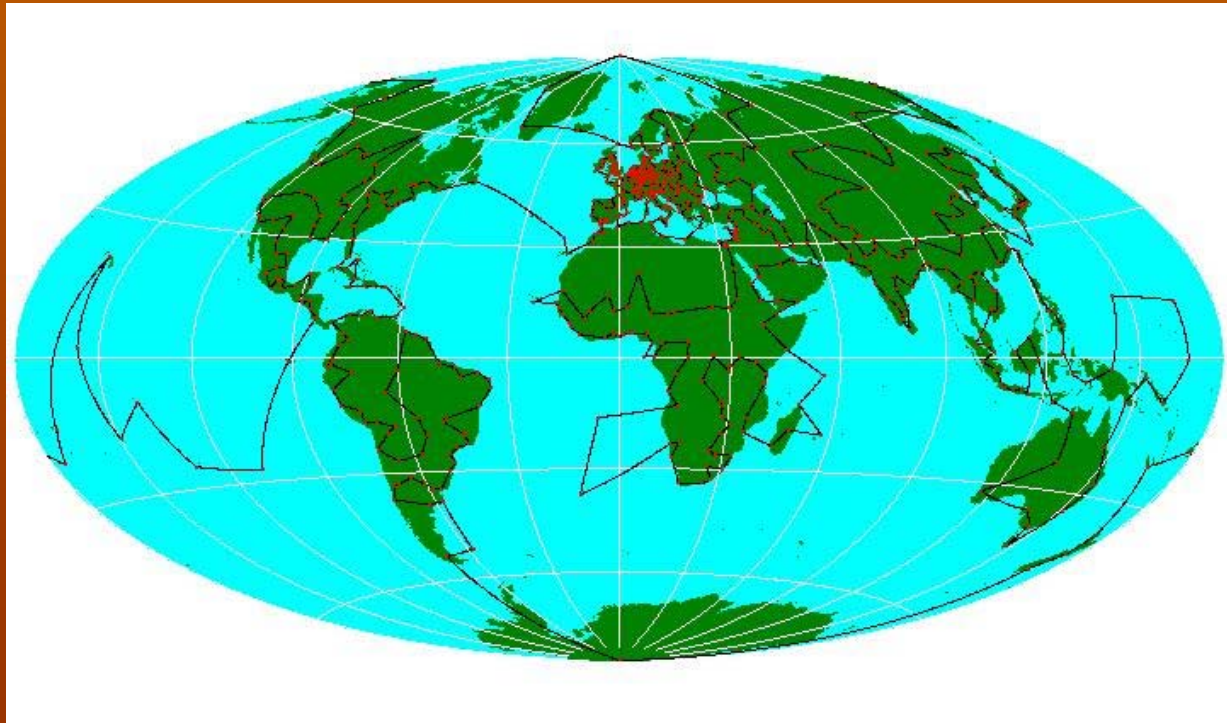


Individuo
Mutado

•
•
•

Problema del viajante

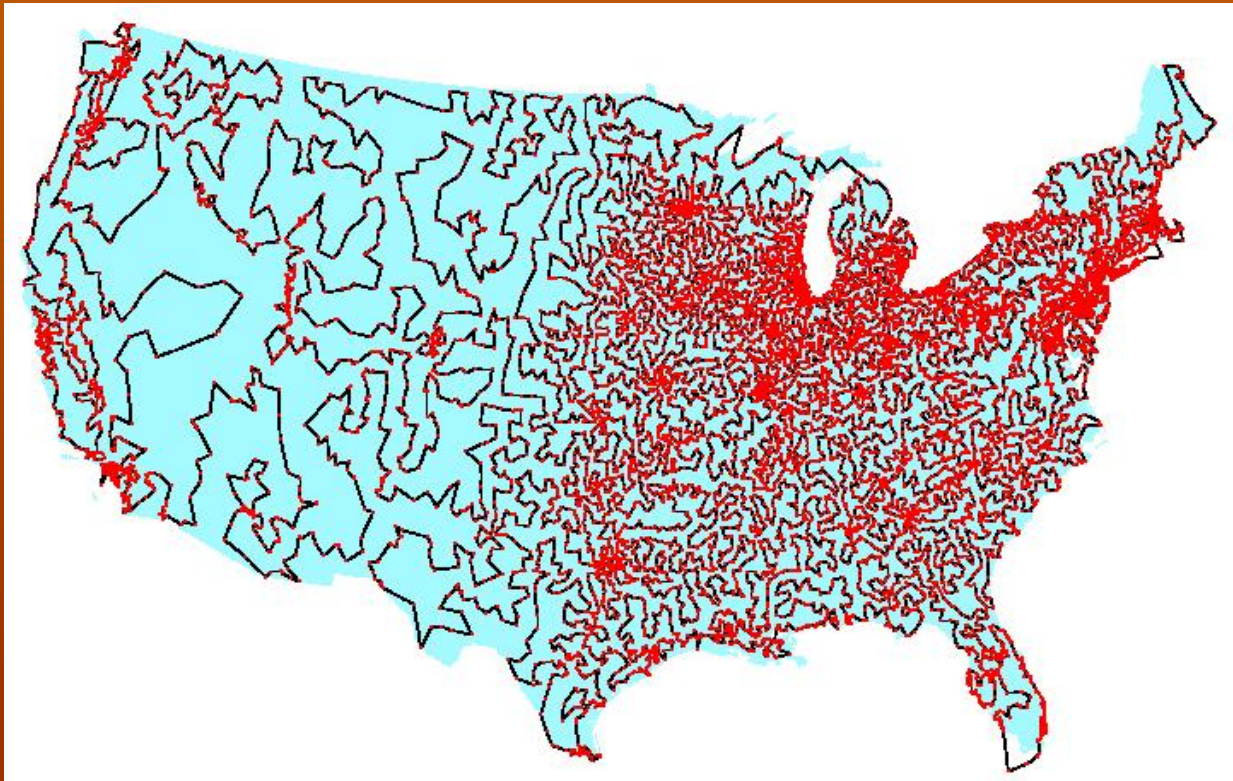
- **1988 Holland y Groetschel encuentran, mediante técnicas evolutivas, la ruta óptima para recorrer 666 lugares de interés en el mundo**



•
•
•

Problema del viajante

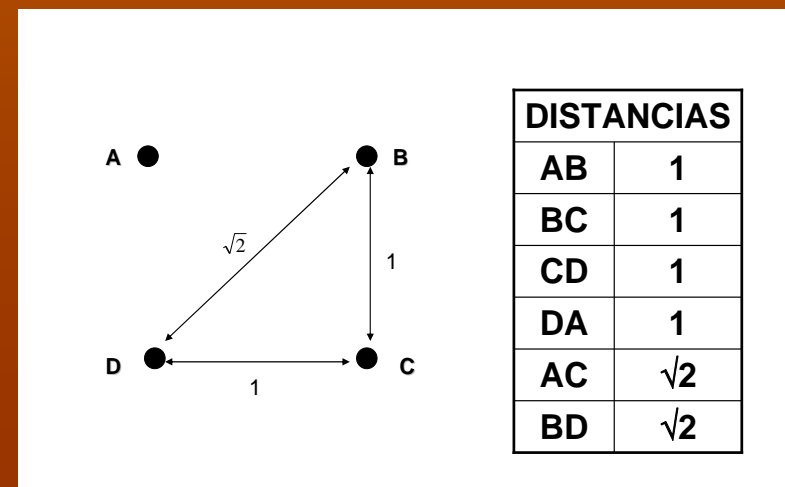
- 1998 se encuentra la ruta para recorrer las 13.509 ciudades de EEUU con más de 500 habitantes



Problema de optimización: El viajante

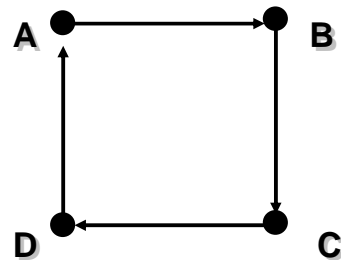
• PROBLEMA DEL VIAJANTE

- Pasos básicos para resolverlo con AGs:
 - 1. Crear una población inicial: grupo de recorridos aleatoriamente.
 - 2. Encontrar los dos mejores en la población y combinarlos.
- A veces el algoritmo llega a un punto donde todos los individuos son iguales. Para que no ocurra esto:
 - Partir de una población inicial grande y
 - Utilizar el método de mutación
- Ejemplo: Se consideran 4 ciudades:

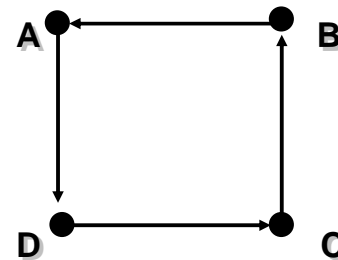


Problema de optimización: El viajante

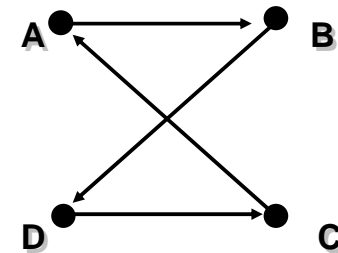
– Posibles combinaciones:



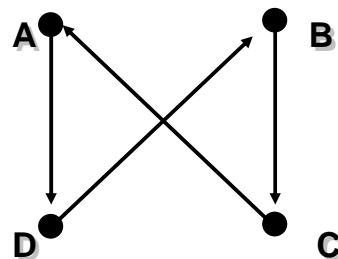
Viaje 1
Distancia = 4



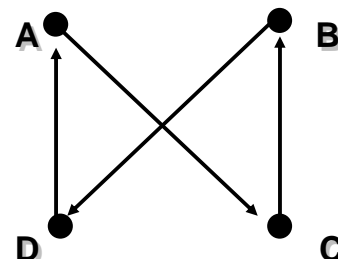
Viaje 2
Distancia = 4



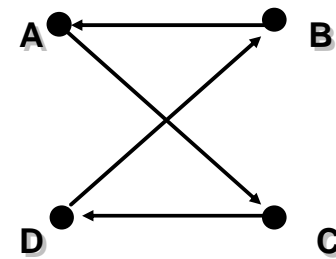
Viaje 3
Distancia = $2+2\sqrt{2}$



Viaje 4
Distancia = $2+2\sqrt{2}$



Viaje 5
Distancia = $2+2\sqrt{2}$



Viaje 6
Distancia = $2+2\sqrt{2}$

-
-
-

Problema de optimización: El viajante

- **Representación de un individuo:**

INSTANCIA	TRAMO AB	TRAMO BC	TRAMO CD	TRAMO DA	TRAMO AC	TRAMO BD
w_1	bit ₅	bit ₄	bit ₃	bit ₂	bit ₁	bit ₀

- **La función de evaluación debe cumplir:**

- Número de tramos recorridos igual a 4.

$$Fitness(w)_1 = 1 - \frac{\left| C - \sum_{i=0}^{L-1} b_i \right|}{C}$$

C: número de ciudades

L: longitud de la cadena de bits

- La distancia recorrida debe ser mínima

$$Fitness(w)_2 = \frac{\sum_{i=0}^{L-1} b_i T_i}{\sum_{i=0}^{L-1} b_i}$$

T_i: distancia del tramo i

- Ecuación de aptitud de cada individuo:

$$Fitness(w) = Fitness(w)_1 - 0.1 * Fitness(w)_2$$

-
-
-

Red de Neuronas Artificiales (RNA)

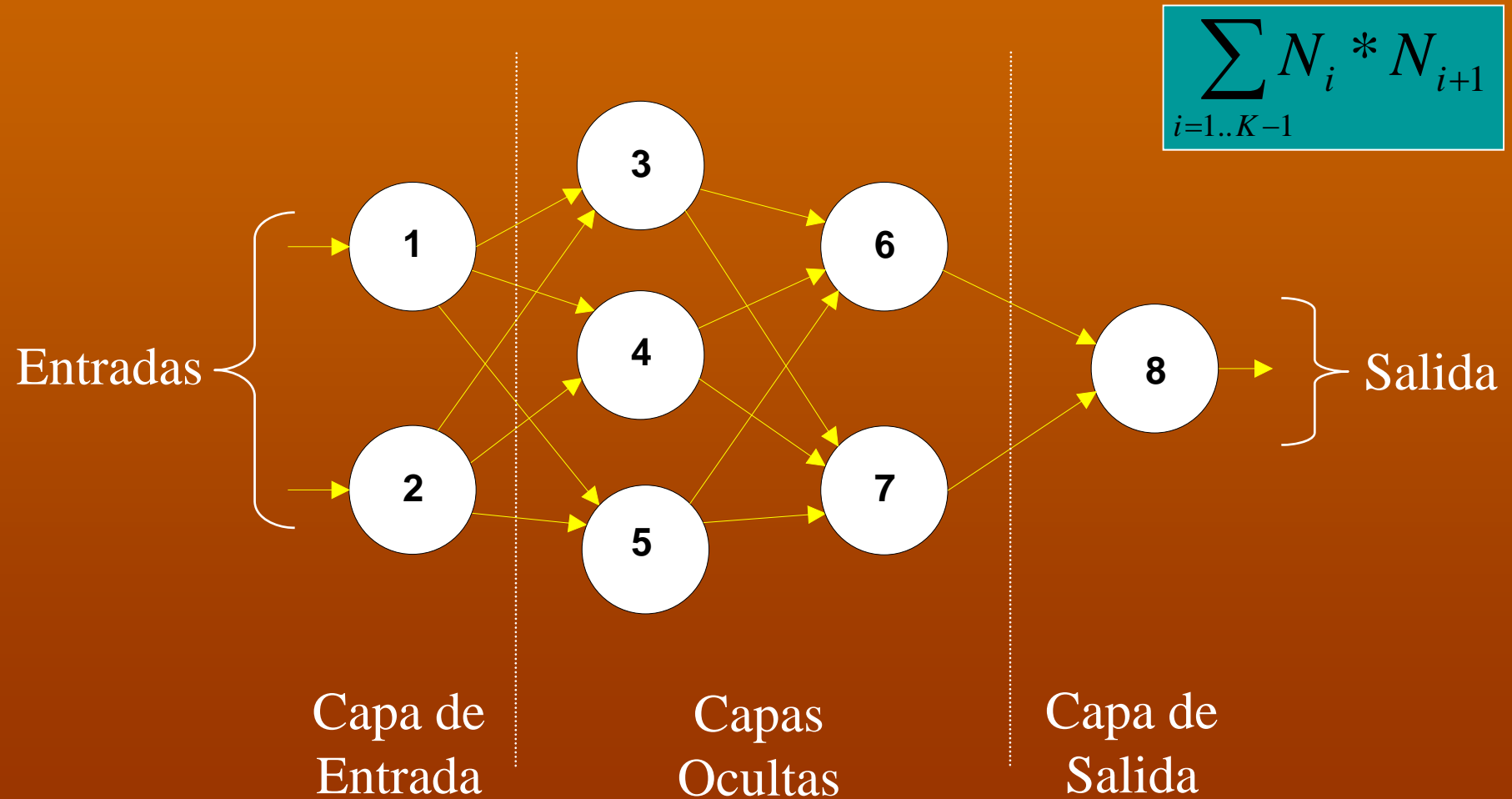
Una RNA se puede definir como un modelo artificial basado en la conexión de varios elementos de proceso (EP) (neuronas) para que conjuntamente realicen una función común.

Su funcionamiento interno es muy poco conocido.

La idea básica es encontrar la topología idónea de una RNA que resuelva una determinada tarea.

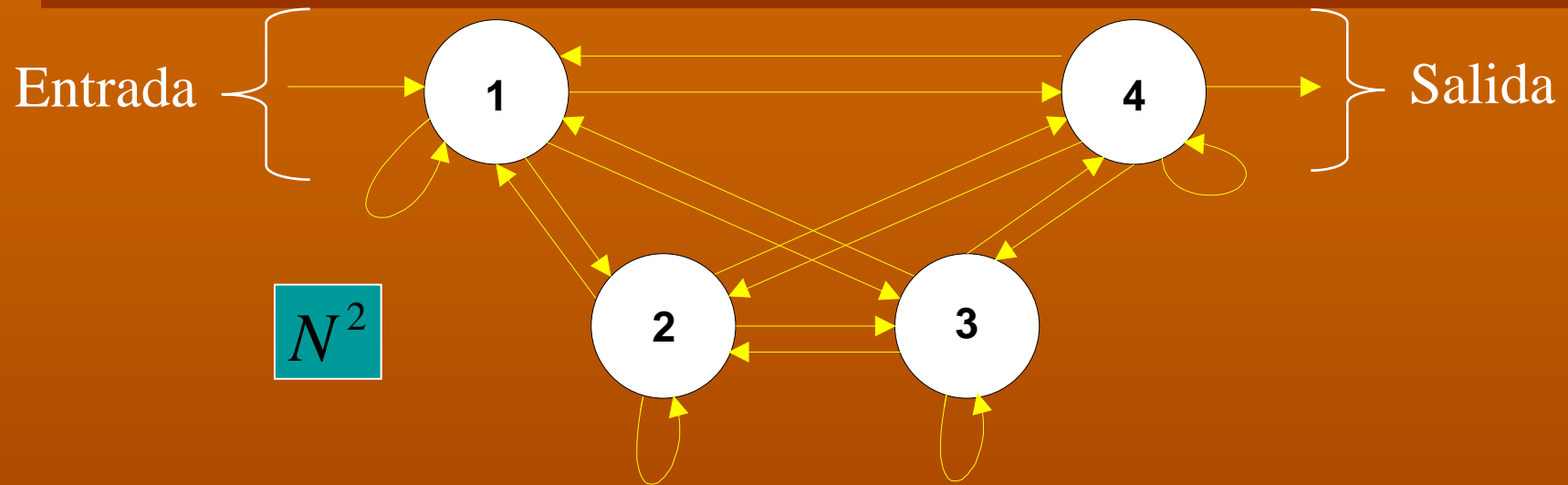
Por topología se entiende organización, número de neuronas y conectividad entre neuronas.

Estructura de una RNA no Recurrente



•
•
•

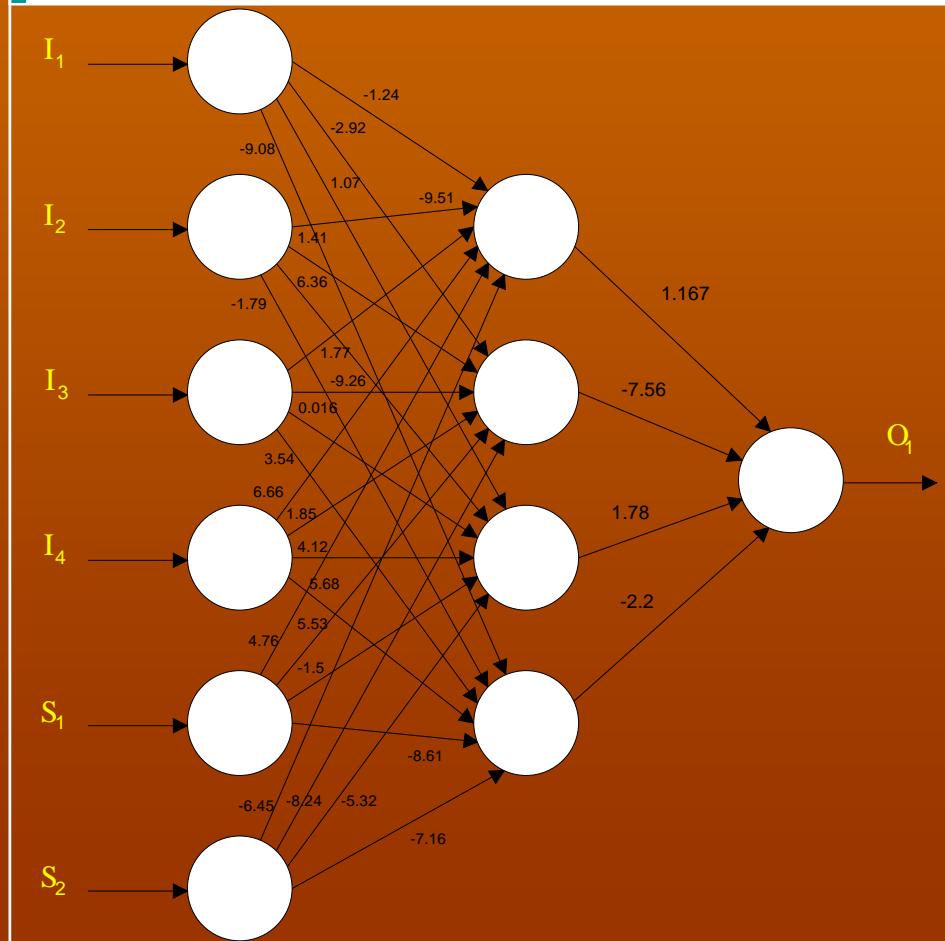
Estructura de una RNA Recurrente



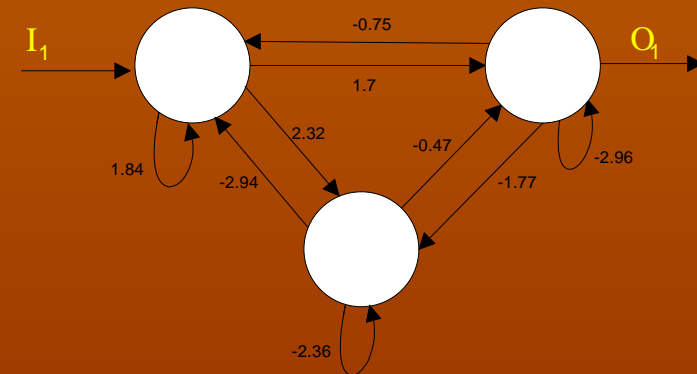
- No existe ninguna restricción en su conectividad.
- Desaparece el concepto de “capas”.
- Se aplican en tareas de reconocimiento, predicción, producción y asociación de secuencias.
- El aprendizaje de este tipo de redes es complicado y lento.

Ejemplos de RNA

Multiplexor (4 a 1)



Señal de Reloj



-
-
-

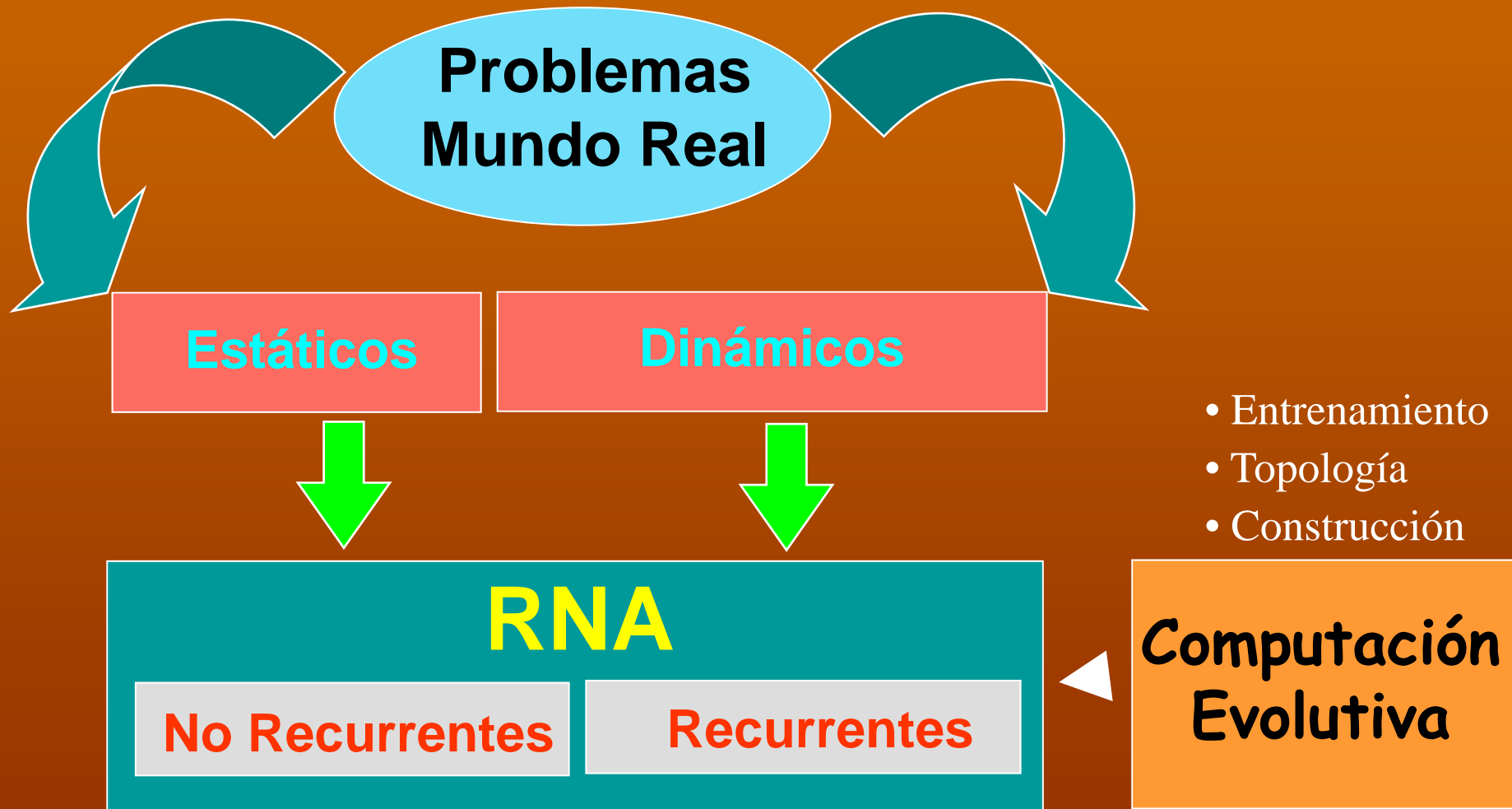
Entrenamiento de RNA

Modelo Supervisado: Partiendo de pares de vectores entrada se calculan unos pesos que produzcan unos vectores de salida lo más parecidos a los deseados

El entrenamiento no es directo (aplicando una "fórmula mágica") sino que partiendo de unos pesos iniciales (normalmente aleatorios y pequeños -respecto a los valores de las entradas) se realizan modificaciones (normalmente bastante pequeñas de forma que se asemeje a un movimiento bastante continuo y no a saltos) hasta alcanzar los pesos que se buscan.

El algoritmo de retropropagación utiliza como función de error una suma de cuadrados (que viene a ser el cuadrado de la distancia euclídea).

Bloques Constructivos



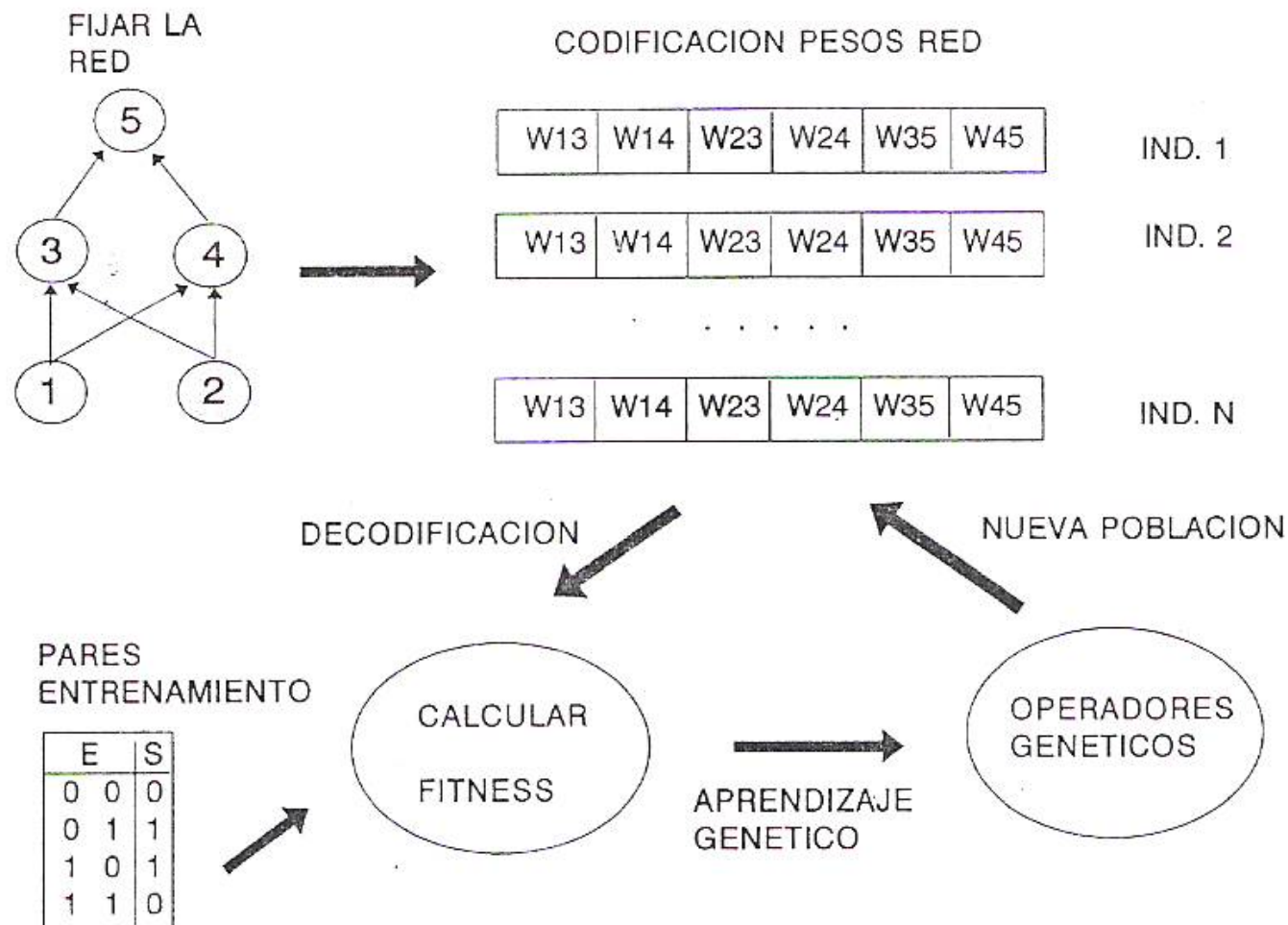
•
•
•

Diseño de RNA

Tres formas de aplicar la computación evolutiva en el diseño de las RNA:

1. **Entrenamiento:** Búsqueda del conjunto óptimo de pesos de conexión
2. **Topología:** Búsqueda del número de neuronas y conectividad óptimos
3. **Construcción:** Utilización de CE para obtener reglas de construcción de redes

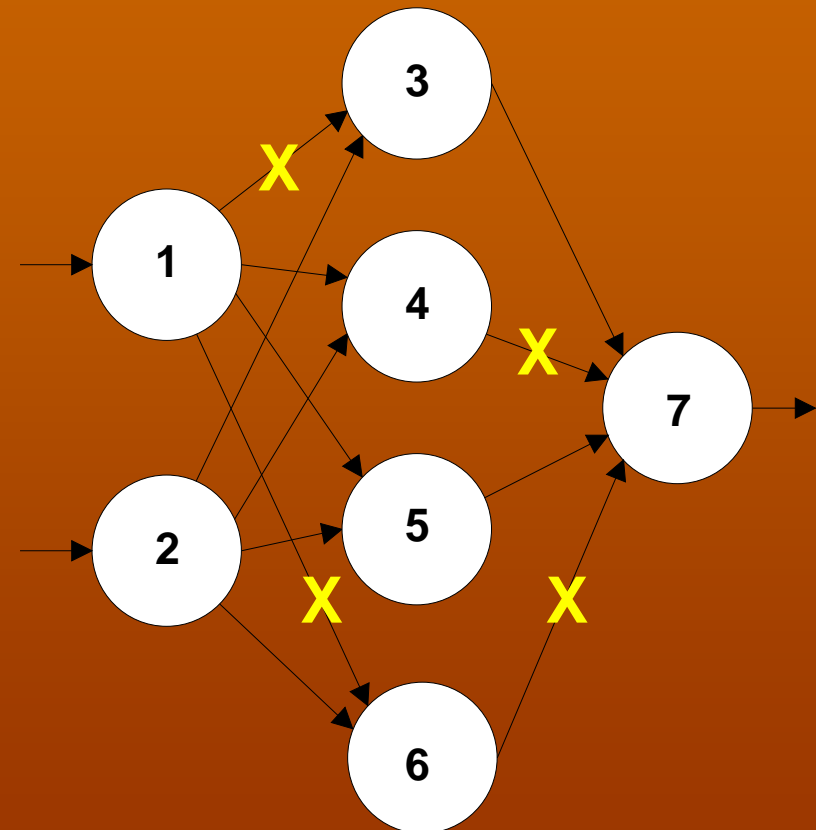
Entrenamiento: Optimización de pesos



...

Codificación de Conectividad

Neurona de Entrada	Neurona de Salida						
	1	2	3	4	5	6	7
	1		0	1	1	0	
	2		1	1	1	1	
	3						1
	4						0
	5						1
	6						0
	7						



-
-
-

Entrenamiento: Optimización de pesos

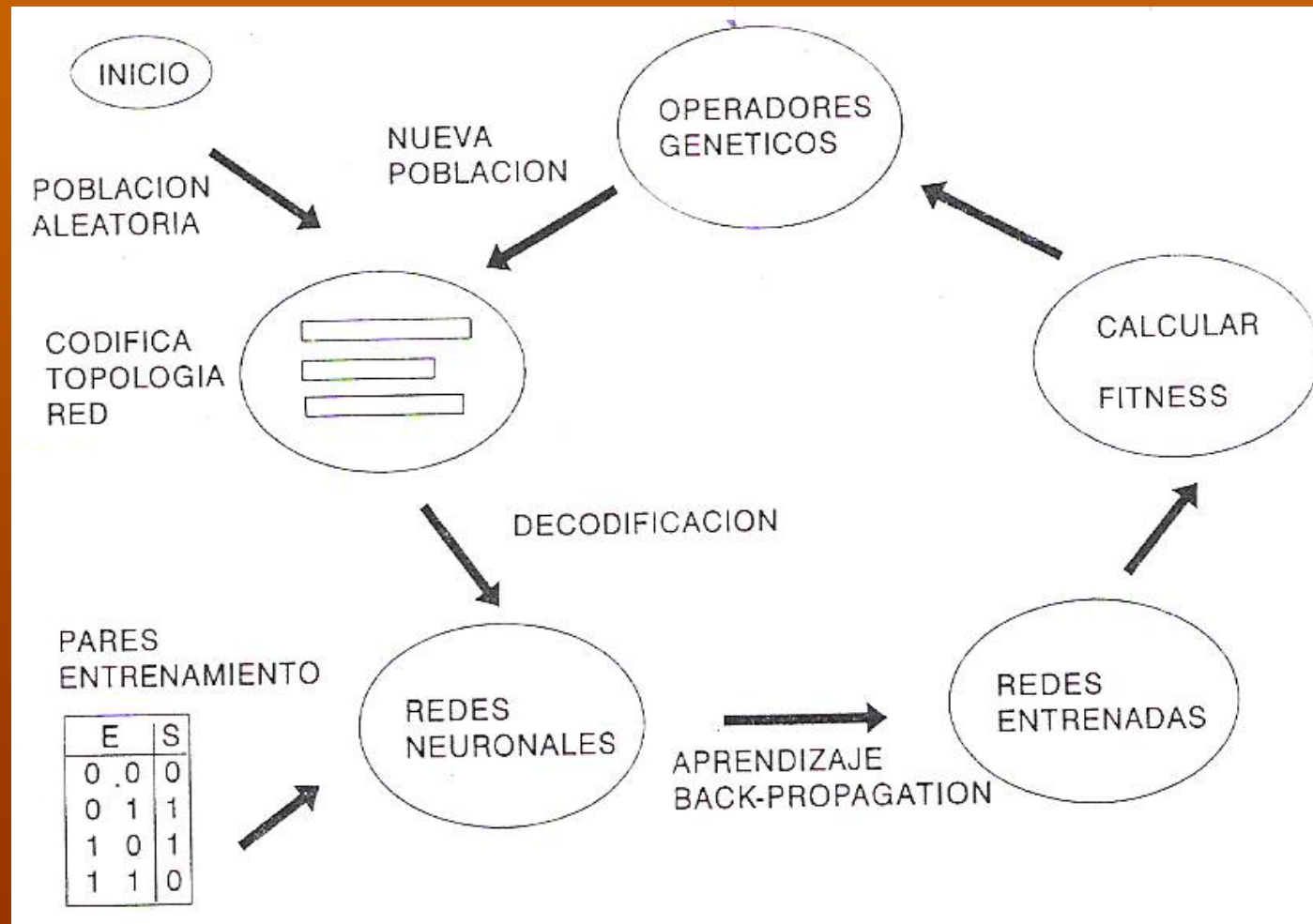
- **Ventajas**

- Sirve para cualquier tipo de RNA
 - Funcionamiento de las neuronas
 - Conexión hacia delante y recurrente

- **Función de evaluación**

- Añadir una penalización a la función de evaluación
 - $\text{fitness} = \text{ECM} + n^{\circ} \text{ conex no } 0$

Búsqueda de la topología



•
•
•

Búsqueda de la topología

- **Problemática**

- Cálculo del error
 - Tiempo de cálculo
 - Restringir por ECM mínimo o por ciclos de aprendizaje
 - Topologías distintas aprenden a distinta velocidad
 - Inicializaciones distintas generan entrenamientos distintos

- **Codificación de la topología de la red**

- Directa
 - Correspondencia entre los genes y los elementos de la RNA
- Indirecta
 - El individuo indica cómo construir la RNA
- Objetivos
 - Número de capas, neuronas por capa y conectividad entre neuronas

•
•
•

Búsqueda de la topología

- **Codificación de la topología de la red**

- Directa

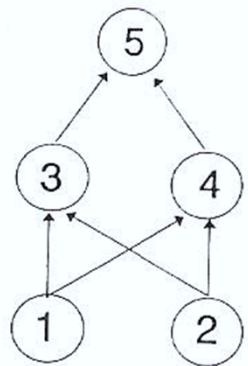
- AAGG

- Array de conexiones

- Hacia delante

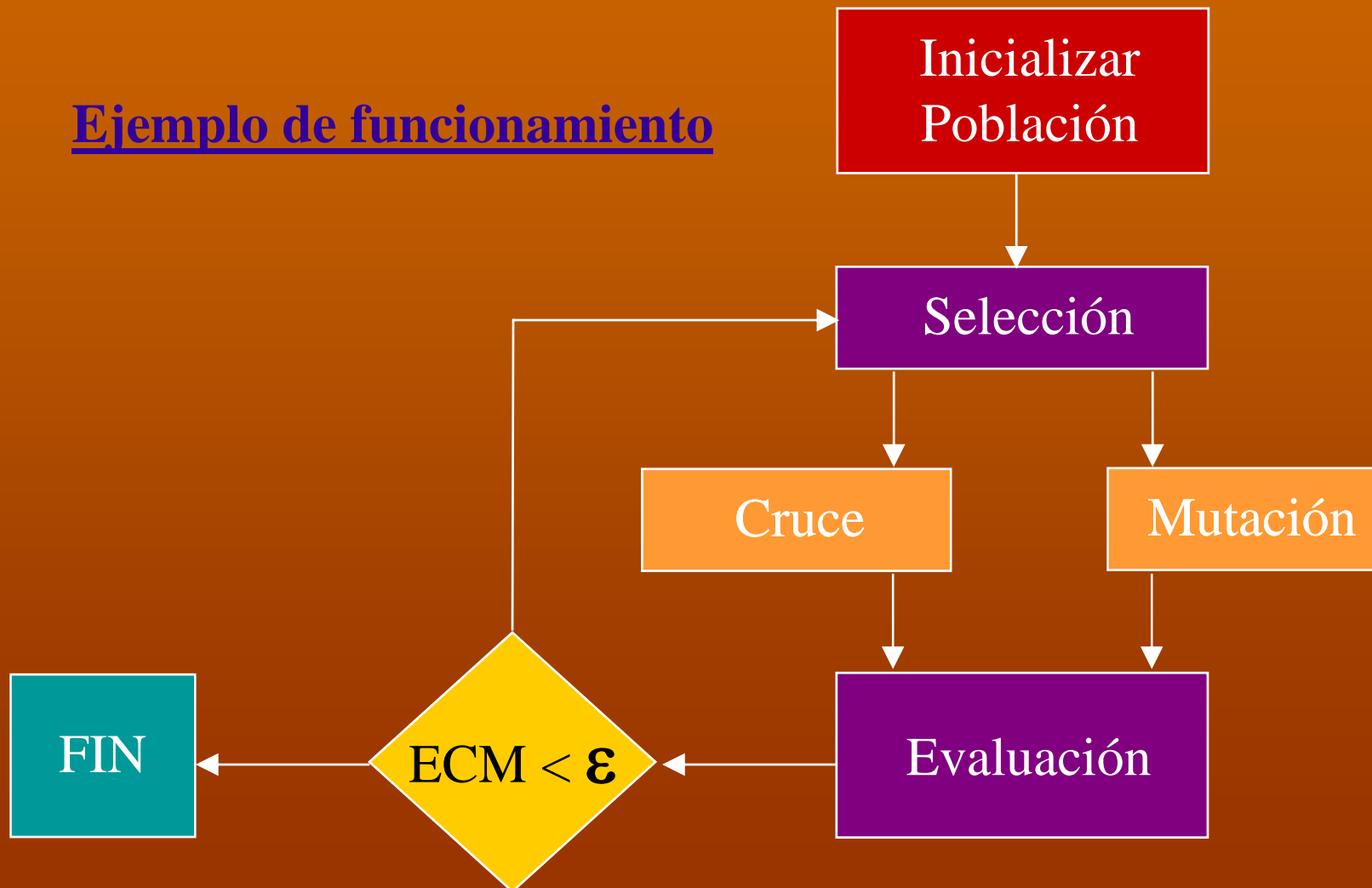
- Recurrente

- Función de evaluación con número de unos

RED	MATRIZ DE CONEXIONES	CODIFICACION DIRECTA																																										
	<table><tr><th>DESDE</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th></tr><tr><th>A</th><td></td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>2</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>3</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><td>4</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><td>5</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td></tr></table>	DESDE	1	2	3	4	5	A						1	0	0	0	0	0	2	0	0	0	0	0	3	1	1	0	0	0	4	1	1	0	0	0	5	0	0	1	1	0	00000000000110001100000110 ⋮
DESDE	1	2	3	4	5																																							
A																																												
1	0	0	0	0	0																																							
2	0	0	0	0	0																																							
3	1	1	0	0	0																																							
4	1	1	0	0	0																																							
5	0	0	1	1	0																																							

Estructura del Algoritmo Genético

Ejemplo de funcionamiento



Problema de satisfacción: FNC

- **FORMAS NORMALES CONJUNTIVAS DE LA LÓGICA**

- Secuencia de cláusulas unidas por la relación AND. Las cláusulas son una disyunción de literales.
- Satisfacibilidad: valores de verdad de los literales que evalúen la secuencia a *true*
- Representación: cadena de 6 bits indicando el valor *true* o *false* de cada literal. Ej:

a	b	c	d	e	f
1	0	1	0	1	0

- Se requiere que los operadores genéticos generen descendientes que evalúen a cierta la expresión

Problema de satisfacción: FNC

- Una posible solución con el operador selección y la función fitness:
 - La expresión estará formada por N cláusulas y se representará como una cadena de N bits
 - Se añadirá un bit, que será la función fitness, con rango de 0 a N, que representa el número de cláusulas que satisfacen los literales.
 - Ejemplo: Para la expresión:

$$(\neg a \vee c) \wedge (\neg a \vee c \vee \neg e) \wedge (\neg b \vee c \vee d \vee \neg e) \wedge (a \vee \neg b \vee c) \wedge (\neg e \vee f)$$

1 1 0 0 1 0 tiene fitness 1(cláusula 4)

0 1 0 0 1 0 tiene fitness 2 (cláusulas 1 y 2)

0 1 0 0 1 1 tiene fitness 3 (cláusulas 1, 2 y 5)

1 0 1 0 1 1 tiene fitness 5 y es la solución

•
•
•

Conclusiones finales

- DIFERENCIAS CON MÉTODOS TRADICIONALES DE BÚSQUEDA Y OPTIMIZACIÓN:
 - Trabajan con un conjunto de parámetros codificados y no con los parámetros mismos
 - Inician la búsqueda desde un conjunto de puntos, no de uno solo.
 - Usan una función a optimizar en lugar de la derivada u otro conocimiento adicional tan difícil a veces de conseguir.
 - Usan reglas de transición probabilísticas no determinísticas

-
-
-

Conclusiones finales

- VENTAJAS:
 - No necesitan conocimientos específicos sobre el problema que intentan resolver
 - Operan de forma simultánea con varias soluciones. No necesitan recorrer todo el espacio de búsqueda
 - Robustez: el AG encuentra de forma eficiente la solución del problema aunque varíen sus parámetros
 - Algoritmo genético es independiente del problema
 - Se usan para problemas de optimización -maximizar una función objetivo- resultan menos afectados por los máximos locales
 - Resulta sumamente fácil ejecutarlos en las modernas arquitecturas masivamente paralelas.

•
•
•

Conclusiones finales

- Inconvenientes:
 - Usan operadores probabilísticos,
 - Pueden tardar mucho en converger, no converger en absoluto o incluso converger prematuramente.
- Cuando NO usar AG:
 - Si se requiere forzosamente llegar al optimo global
 - Si conocemos la función de optimización
 - Si el problema esta muy delimitado y se presta a un tratamiento analítico (funciones de una variable)
 - Si la función es suave y convexa

•
•
•

Referencias Bibliográficas

Holland, J.H. “*Adaptation in natural and artificial systems*”,
Ann Arbor: The University of Michigan Press, 1975.

Goldberg, D.E. “*Genetic Algorithms in Search, Optimization
and Machine Learning*”, Addison-Wesley, 1989.

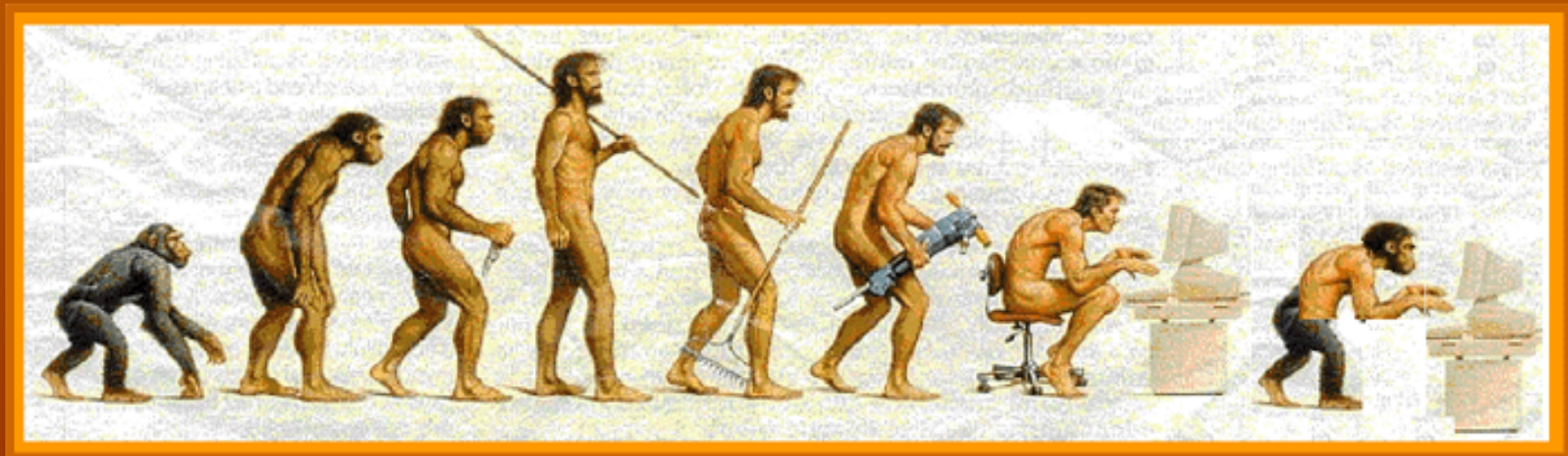
Minsky, M. y Papert, S. “*Perceptrons: An introduction to
computational geometry*”, Cambridge, MA, MIT Press,
1988.

Angeline P.J., Saunders G.M., Pollack J.B. “*An Evolutionary
Algorithm that Constructs Recurrent Neural Networks*”

-
-
-

Computación Evolutiva

PROGRAMACIÓN GENÉTICA



-
-
-

Orígenes

La programación genética surge como una evolución de los algoritmos genéticos tradicionales, manteniendo el mismo principio de selección natural.

Lo que ahora se pretende es resolver los problemas mediante la inducción de programas y algoritmos que los resuelvan





Orígenes de la PG

John Koza 1992 empleó AG para desarrollar programas que hacían ciertas tareas; él las denominó “**genetic programming**” (GP).

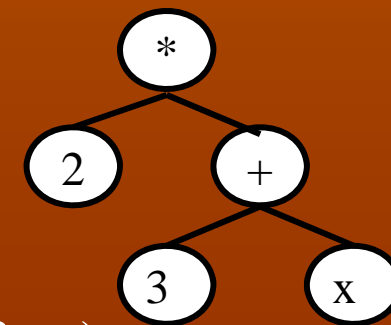
Un método para la creación automática de programas a partir de especificaciones de alto nivel.

•
•
•

Programación Genética

La mayor diferencia entre los algoritmos genéticos y la programación genética es la forma de codificación de la solución al problema

En programación genética la codificación se realiza en forma de árbol, de forma similar a como los compiladores leen los programas según una gramática prefijada



Árbol para la expresión $2*(3+x)$

•
•
•

Programación Genética

La representación en árbol es fácilmente entendible

- Expresión Aritmética

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

- Expresión Lógica

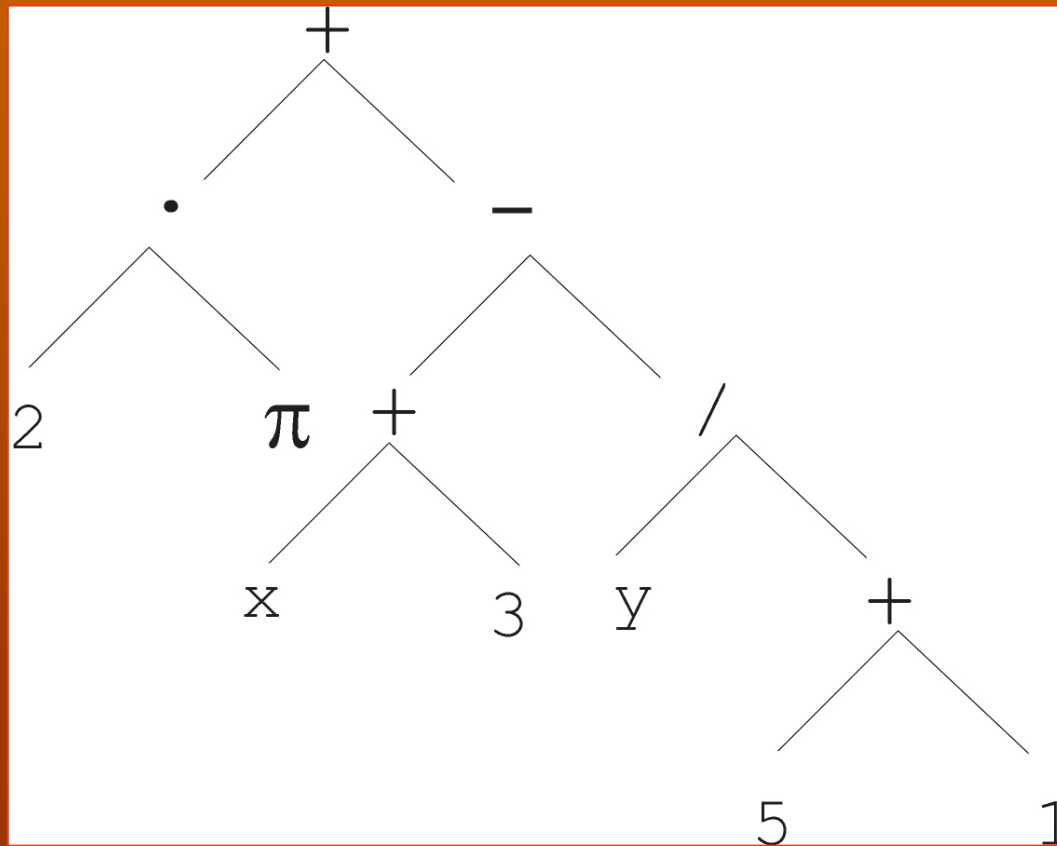
$$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$$

- Programa

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


•
•
•

Expresión Aritmética

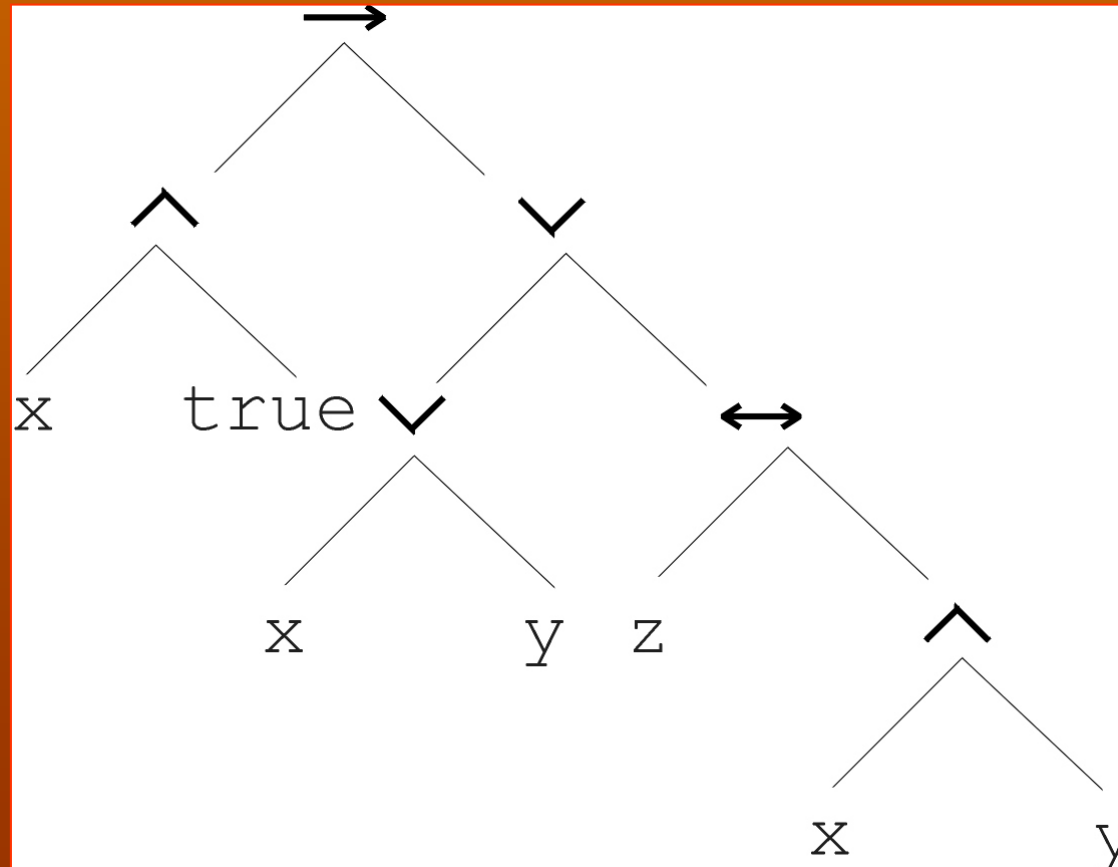


$$2 \cdot \pi + \left((x + 3) - \frac{y}{5 + 1} \right)$$

•
•
•

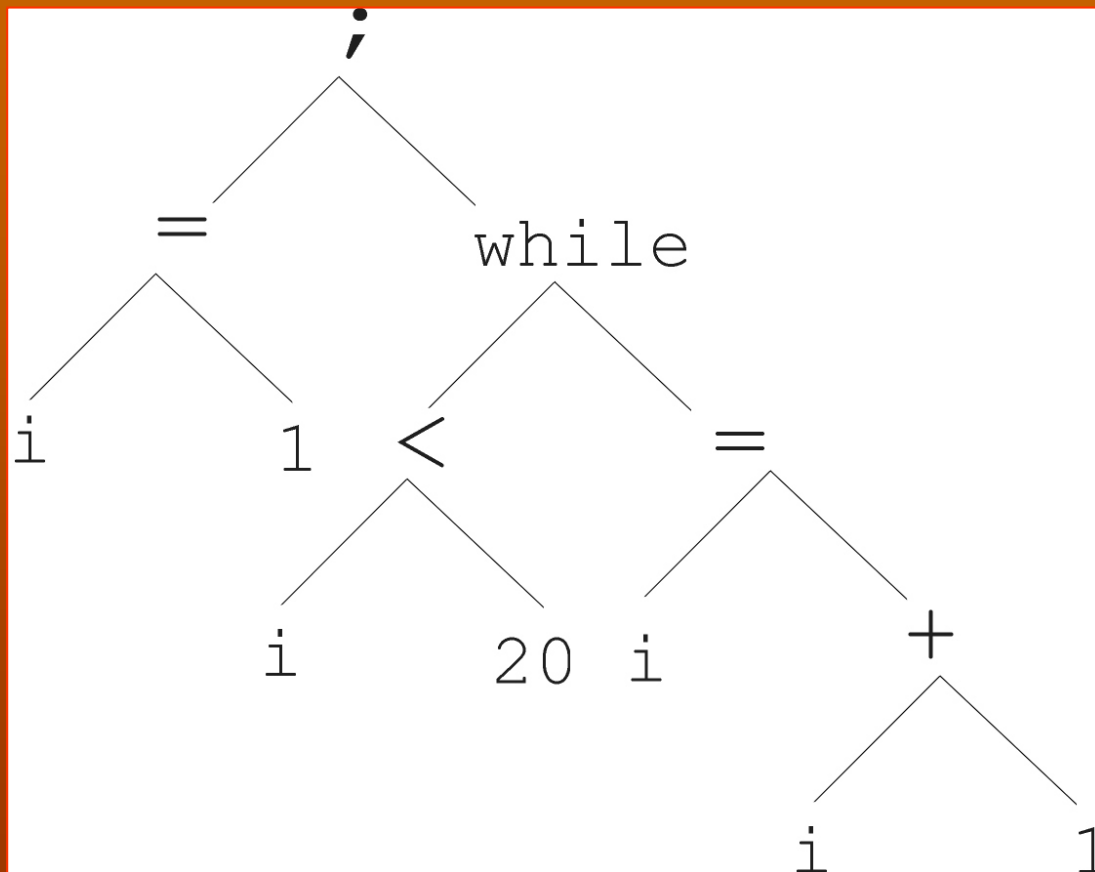
Expresión Lógica

$$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$$



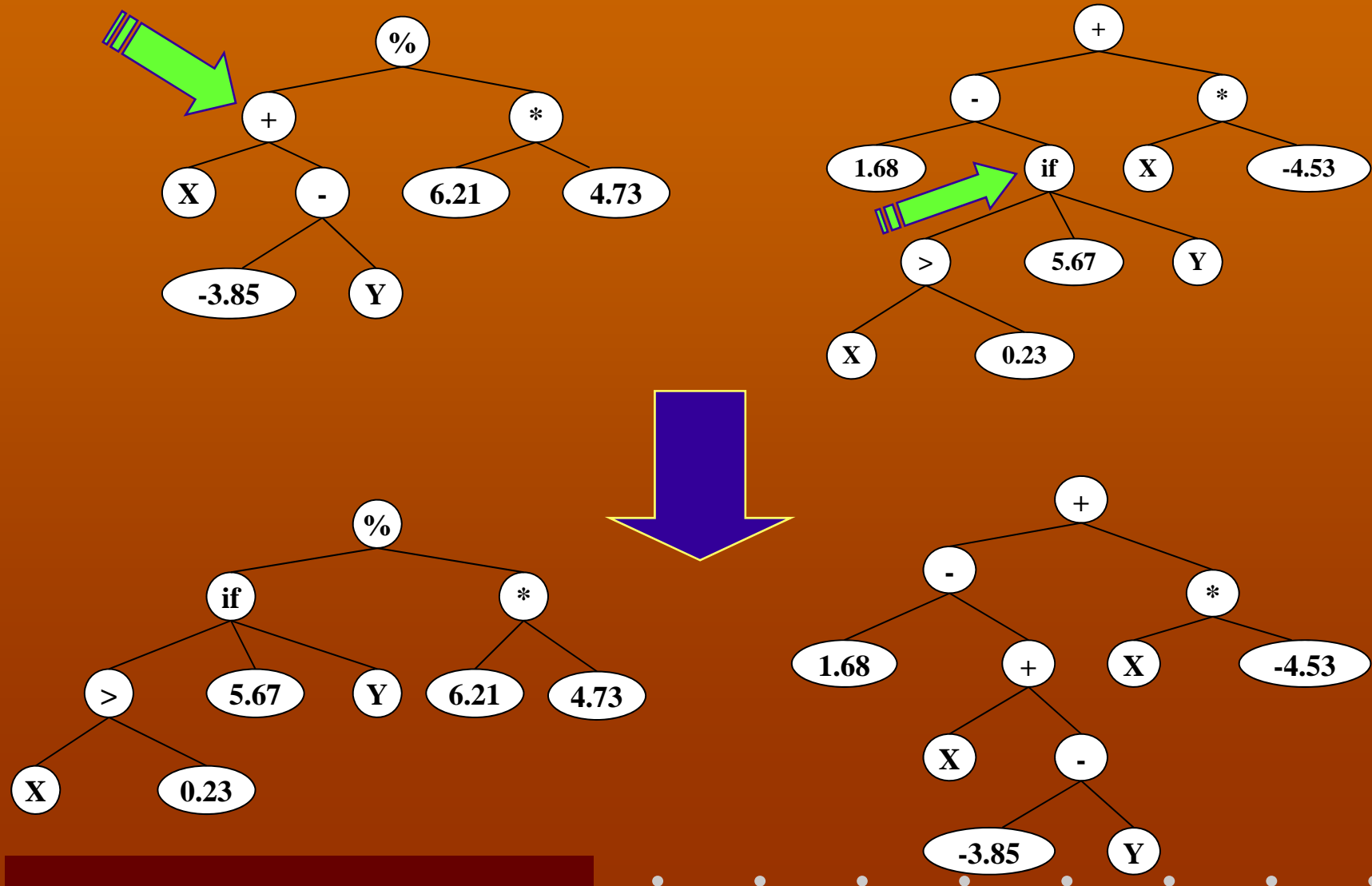
•
•
•

Programa



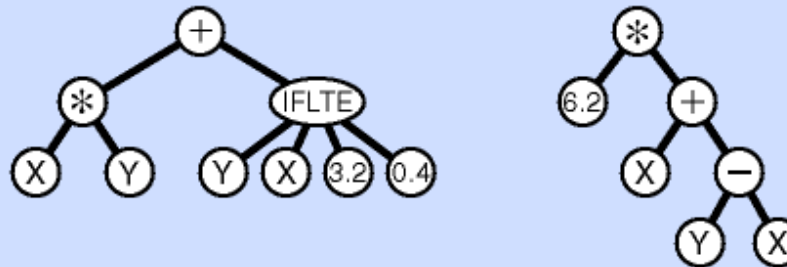
```
i = 1;  
while (i < 20)  
{  
    i = i + 1  
}
```

Operador de Cruce



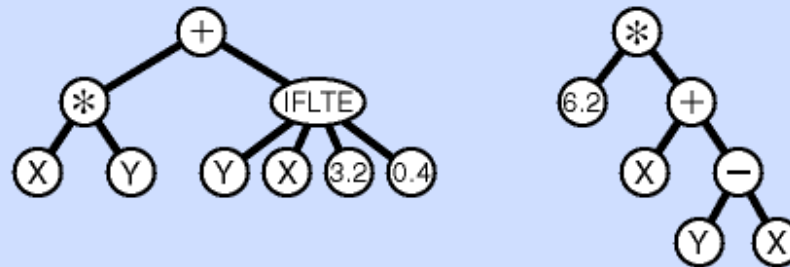
-
-
-

PG: cruce



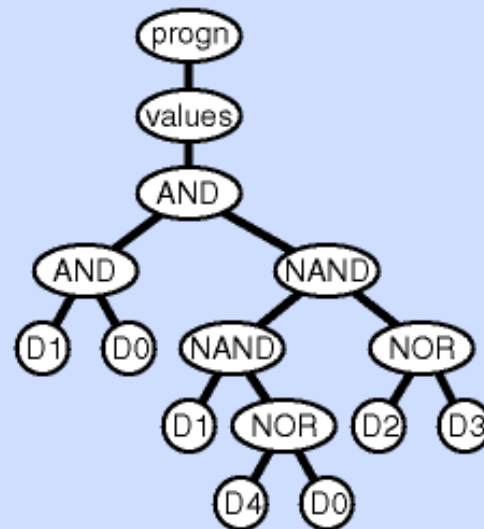
-
-
-

PG: mutación



-
-
-

Programación genética



•
•
•

Programación genética

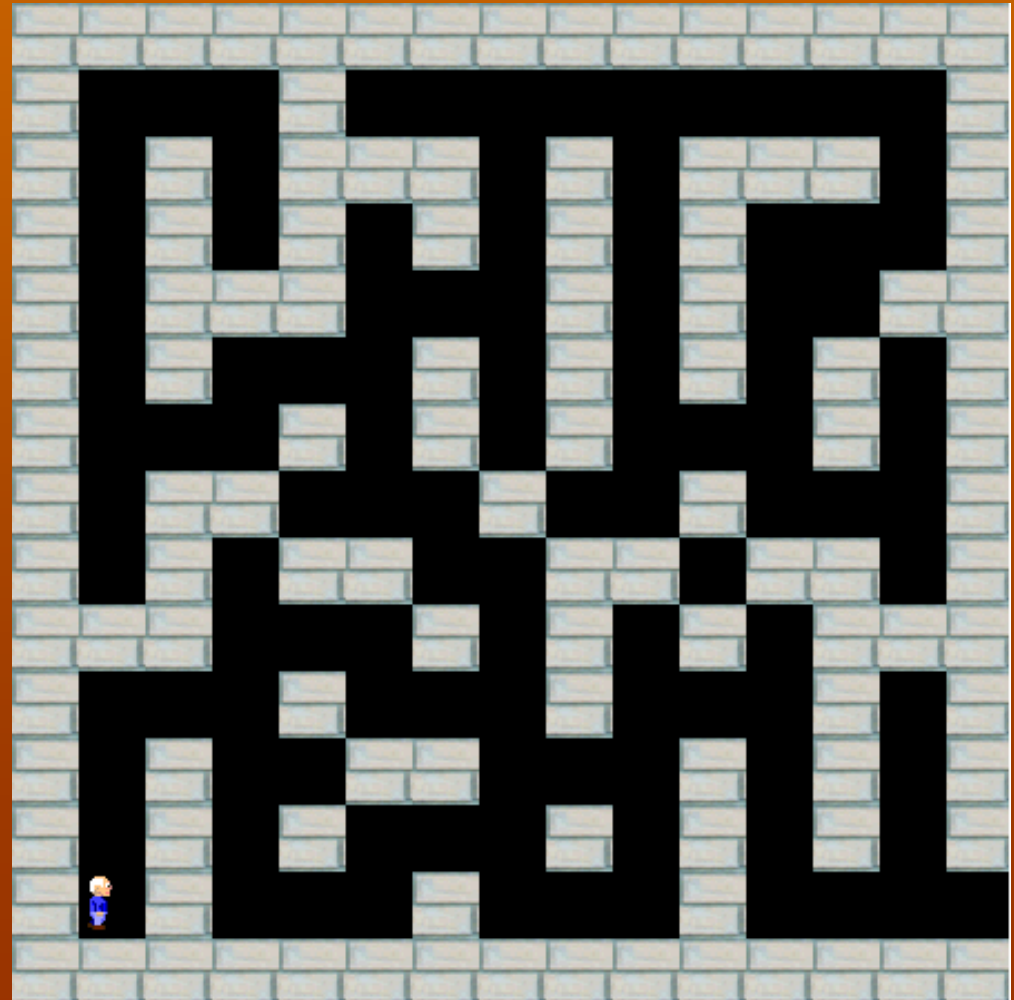
- **Ejemplos de utilización**
 - Regresión simbólica *obtener reglas y ecuaciones de patrones de datos*
 - Desarrollo de programas para computador
 - Sistemas Basados en Reglas
 - El árbol representa una regla
 - Dos ramas
 - rama izquierda representa la parte *if* y
 - rama derecha representa la parte *then*
 - Evaluación frente al problema de las mejores reglas para cada situación

-
-
-

Ejemplo: Laberinto

Un problema de ejemplo es la búsqueda de un algoritmo que ejecutado de forma exhaustiva lleve a la salida de un laberinto

Para ello, y situados en una posición inicial, solo se pueden realizar cuatro tipos de acciones: avanzar hacia delante, girar a la derecha, girar a la izquierda y comprobar si hay una pared delante.



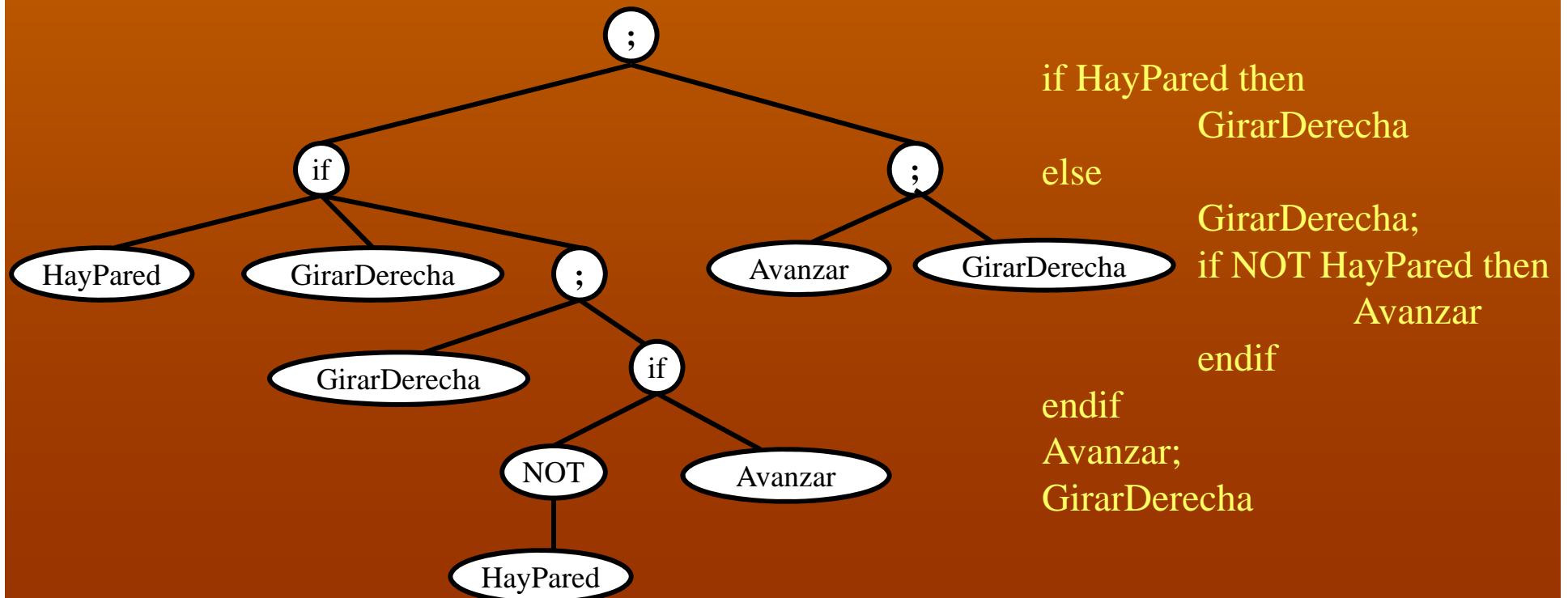
•
•
•

Ejemplo: Laberinto

El conjunto de elementos terminales es el siguiente:

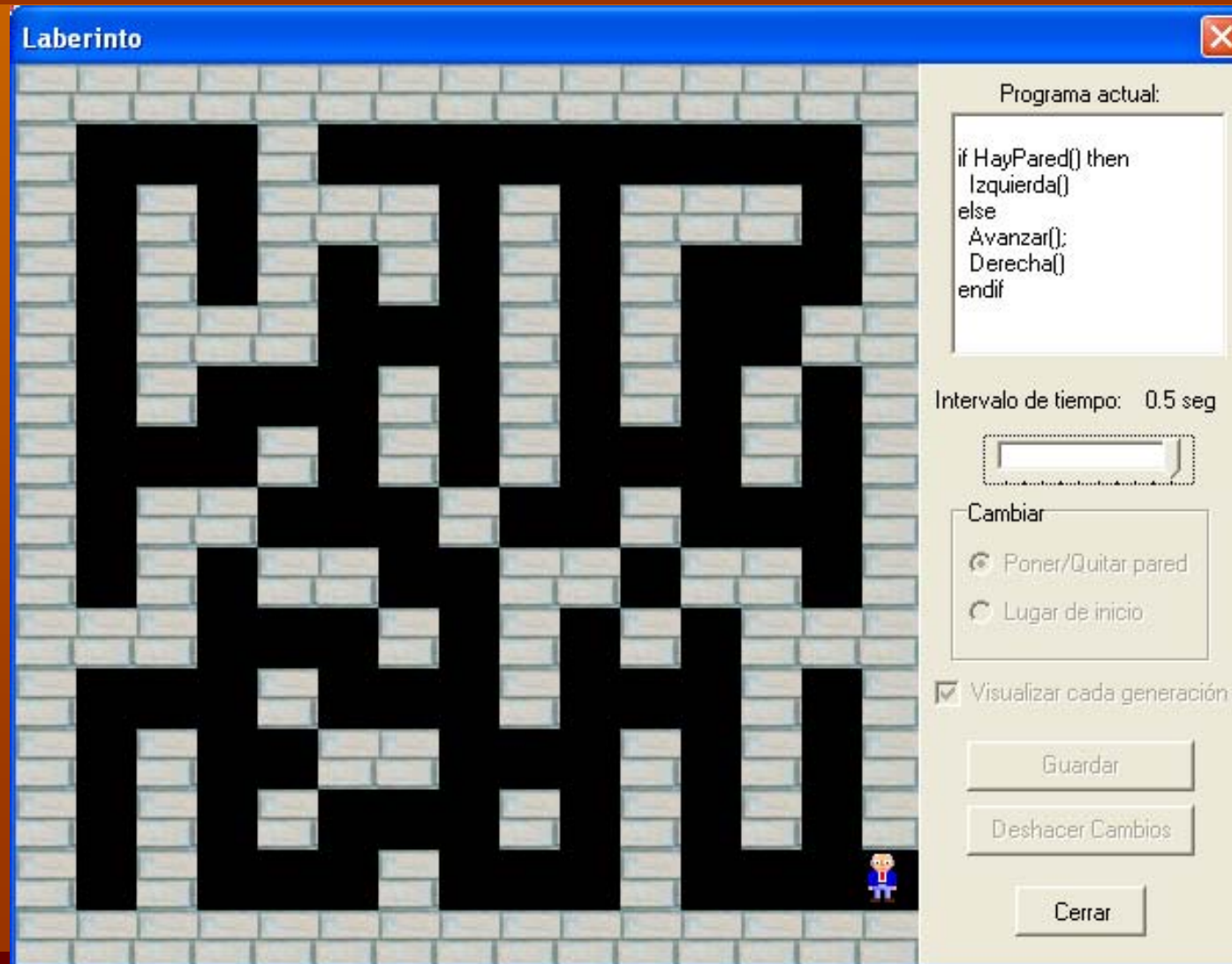
{ Avanzar, GirarDerecha, GirarIzquierda, HayPared }

Un ejemplo de árbol que se puede obtener es:



Ejemplo: Laberinto

Ejemplo



•
•
•

Diseño de RNA

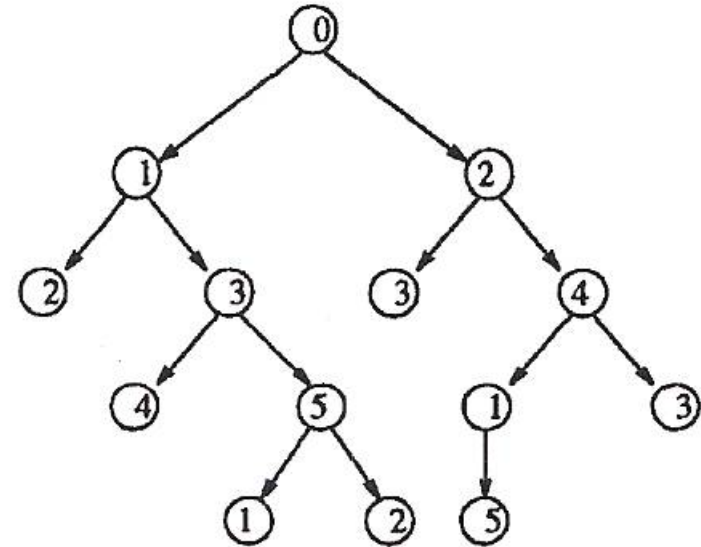
Tres formas de aplicar la computación evolutiva en el diseño de las RNA:

1. **Entrenamiento:** Búsqueda del conjunto óptimo de pesos de conexión
2. **Topología:** Búsqueda del número de neuronas y conectividad óptimos
3. **Construcción:** Utilización de PG para obtener reglas de construcción de redes

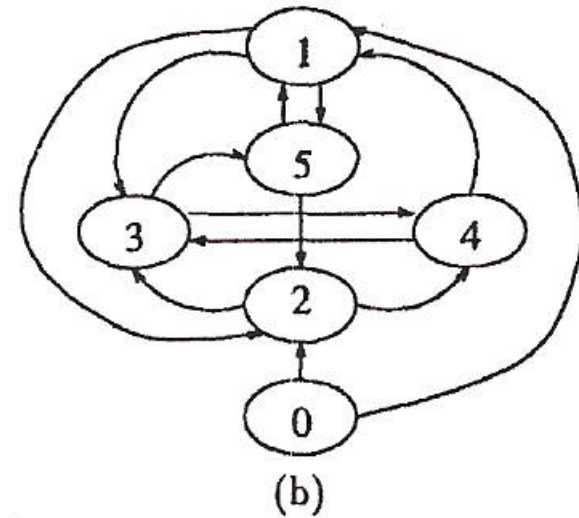
-
-
-

Búsqueda de la topología

- **Codificación de la topología de la red**
 - Directa
 - PG
 - Árbol de nodos y conexiones
 - Función de evaluación ponderando el número de nodos



(a)



(b)

-
-
-

Búsqueda de la topología

- **Codificación de la topología de la red**

- Directa

- PG

- Árbol de nodos y conexiones

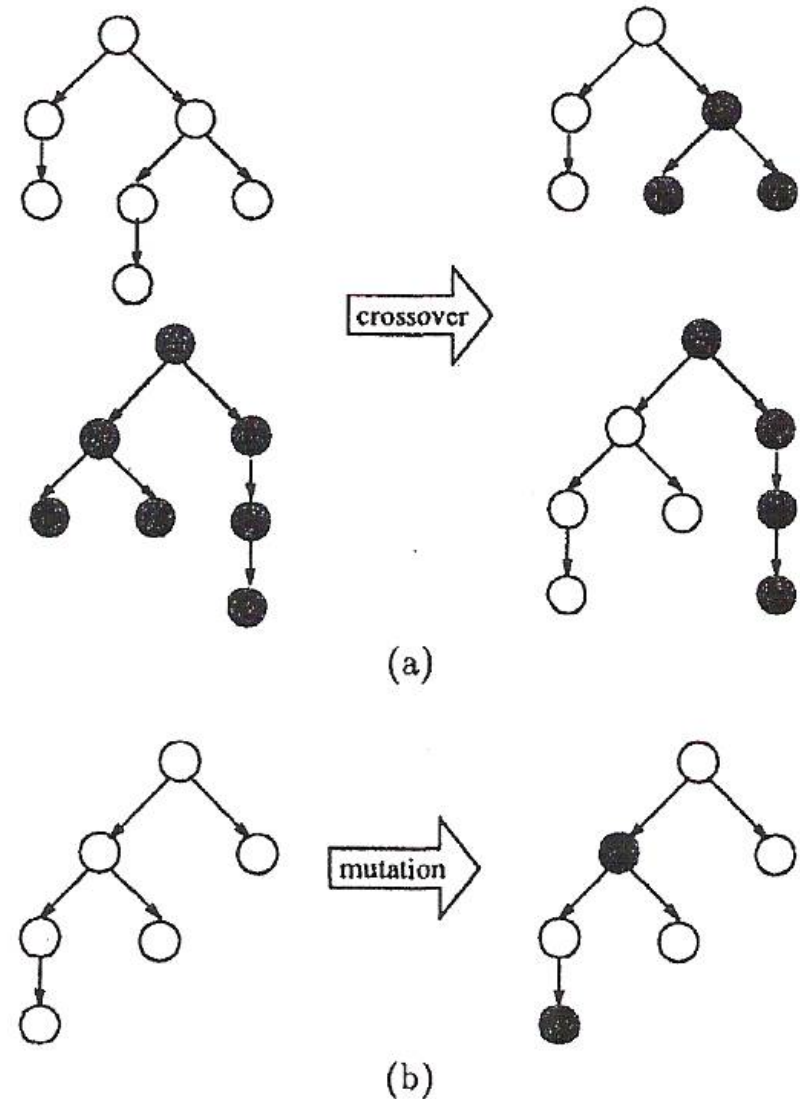
- Operadores

- Cruce

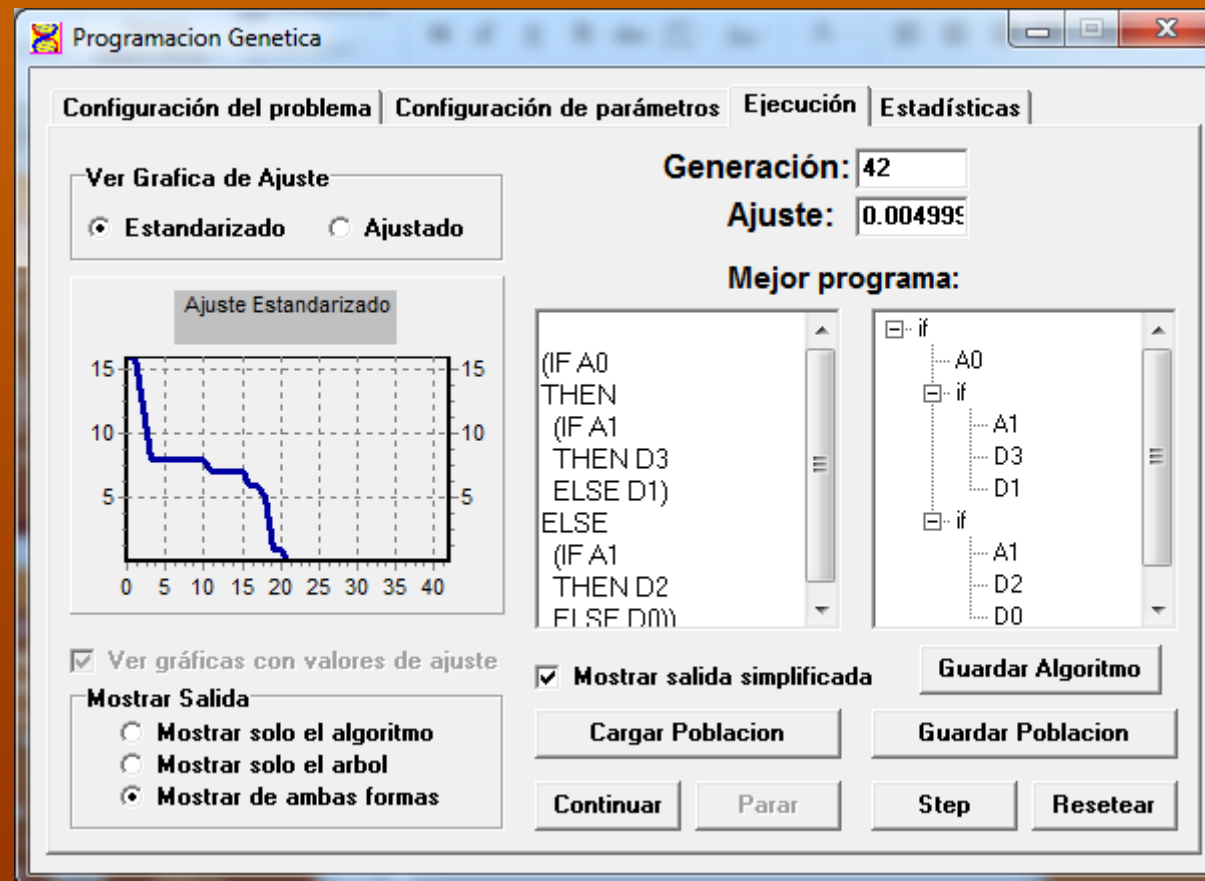
- Mutación

- Cambio de nodo

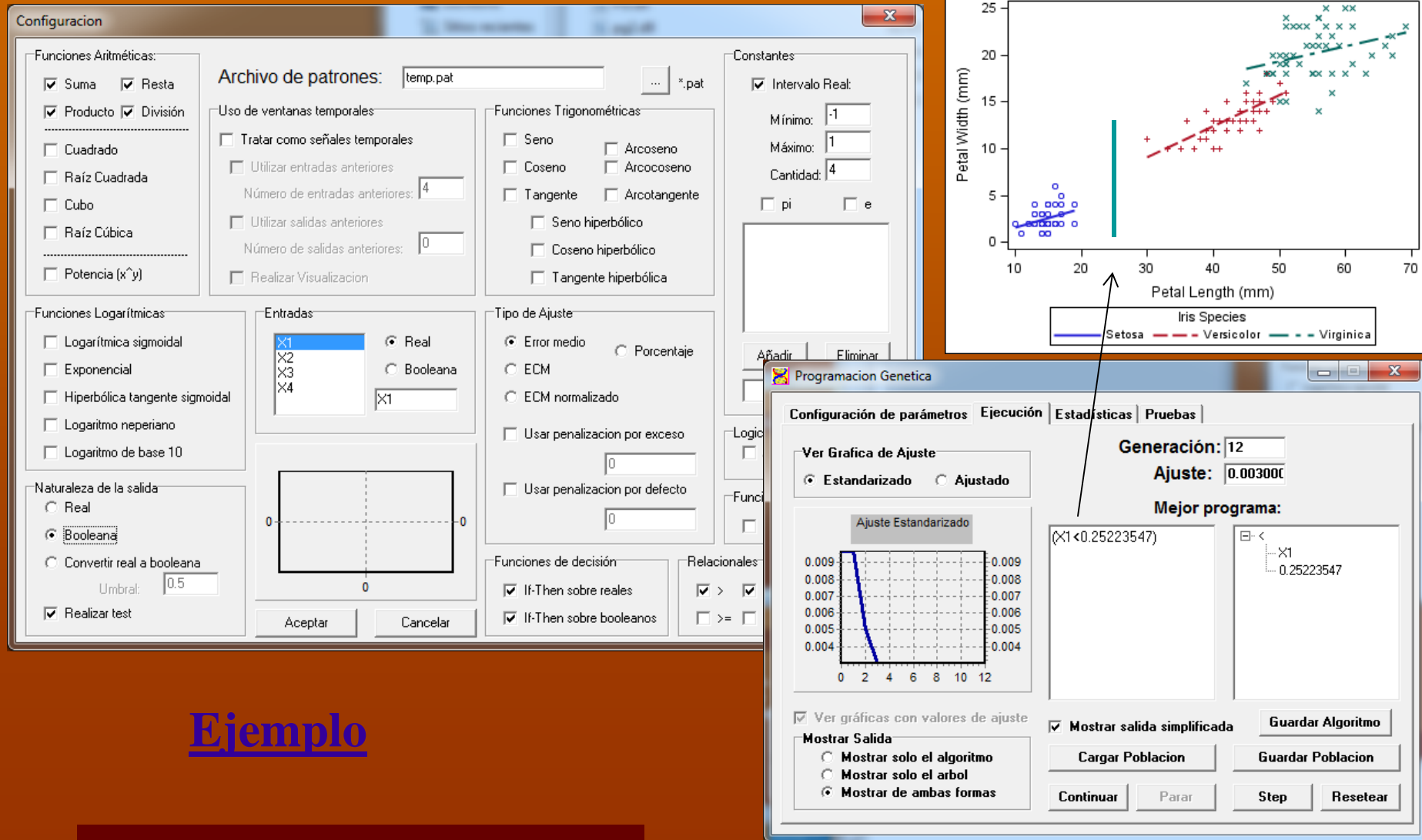
- Inserción y duplicación



Ejemplo: Multiplexor



Ejemplo: Regresión Simbólica



Ejemplo

•
•
•

Conclusiones

- **Naturaleza y biología**
 - Genética: codificación y combinación
 - Ecología: estudio de las poblaciones
- **Comportamientos emergentes**
- **Extracción de reglas**
- **Hardware evolutivo**
 - Robótica



-
-
-

Bibliografía

- **Introducción a la Computación Evolutiva**
 - <http://nas.tic.udc.es/wiki>
- **GA y GP**
 - <http://www.genetic-programming.org/>
 - <http://www.geneticprogramming.com/>
 - Ejemplos web
 - <https://www.obitko.com/tutorials/genetic-algorithms/index.php>