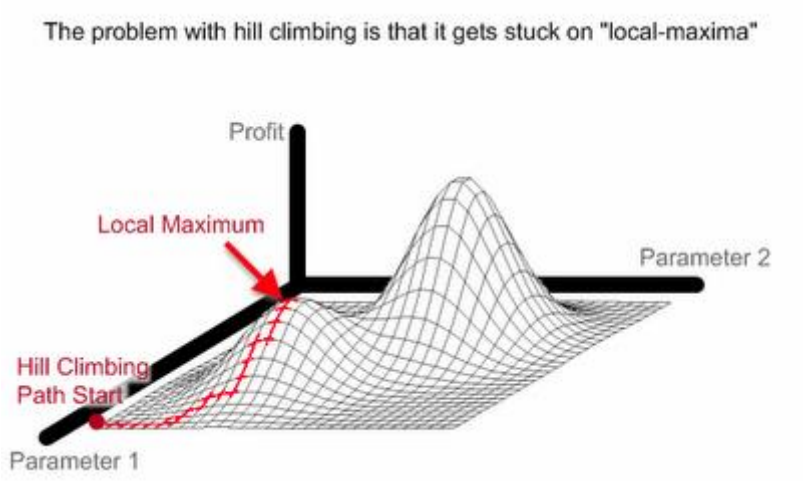
METAHEURÍSTICAS

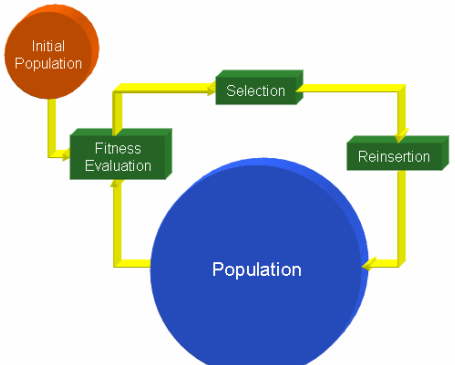
2021 - 2022

- Tema 1. Introducción a las Metaheurísticas
- Tema 2. Modelos de Búsqueda: Entornos y Trayectorias vs Poblaciones
- Tema 3. Metaheurísticas Basadas en Poblaciones
- Tema 4: Algoritmos Meméticos
- Tema 5. Metaheurísticas Basadas en Trayectorias
- Tema 6. Metaheurísticas Basadas en Adaptación Social
- Tema 7. Aspectos Avanzados en Metaheurísticas
- Tema 8. Metaheurísticas Paralelas

(Tema 2) Problemas de la Búsqueda Local. Trayectorias versus Poblaciones

Suele caer en óptimos locales, que a veces están bastante alejados del óptimo global del problema





Se podría tener un modelo que seleccione soluciones, opere con ellas y puedan ser reinsertadas en la población dando lugar a una nueva población. Podemos imitar a la genética, cómo se combinan cromosomas (algoritmos genéticos)

METAHEURÍSTICAS

TEMA 3. METAHEURÍSTICAS BASADAS EN POBLACIONES

Parte I:

1. INTRODUCCIÓN A LA COMPUTACIÓN EVOLUTIVA
2. ALGORITMOS GENÉTICOS
3. ALGORITMOS GENÉTICOS. EXPLORACIÓN VS EXPLOTACIÓN
4. CONCLUSIONES

BIBLIOGRAFÍA
D.B. Fogel (Ed.). Evolutionary Computation. The Fossil Record. (Selected Readings on the History of Evolutionary Computation). IEEE Press, 1998.
A.E. Eiben, J.E. Smith. Introduction to Evolutionary Computation. Springer Verlag 2003.
X-S. Yang, nature inspired metaheuristic algorithms, Luniver Press, 2010

METAHEURÍSTICAS

TEMA 3. METAHEURÍSTICAS BASADAS EN POBLACIONES

Parte II:

1. ALGORITMOS EVOLUTIVOS PARA OPTIMIZACIÓN CONTINUA
2. ALGORITMOS GENÉTICOS PARA OPT. CONTINUA
3. EVOLUCIÓN DIFERENCIAL
4. ESTRATEGIAS DE EVOLUCIÓN
5. NUEVOS MODELOS BIOINSPIRADOS PARA OPTIMIZACIÓN DE PARÁMETROS

BIBLIOGRAFÍA

D.B. Fogel (Ed.). Evolutionary Computation. The Fossil Record. (Selected Readings on the History of Evolutionary Computation). IEEE Press, 1998.

A.E. Eiben, J.E. Smith. Introduction to Evolutionary Computation. Springer Verlag 2003.

METAHEURÍSTICAS

TEMA 3. METAHEURÍSTICAS BASADAS EN POBLACIONES

Parte I:

1. INTRODUCCIÓN A LA COMPUTACIÓN EVOLUTIVA
2. ALGORITMOS GENÉTICOS
3. ALGORITMOS GENÉTICOS. EXPLORACIÓN VS EXPLOTACIÓN
4. CONCLUSIONES

BIBLIOGRAFÍA
D.B. Fogel (Ed.). Evolutionary Computation. The Fossil Record. (Selected Readings on the History of Evolutionary Computation). IEEE Press, 1998.
A.E. Eiben, J.E. Smith. Introduction to Evolutionary Computation. Springer Verlag 2003.

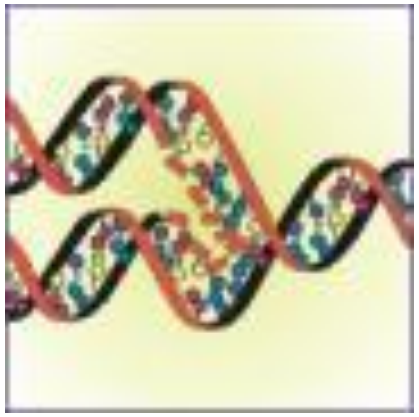
METAHEURÍSTICAS

TEMA 3. METAHEURÍSTICAS BASADAS EN POBLACIONES

INTRODUCCIÓN A LA COMPUTACIÓN EVOLUTIVA

1. INTRODUCCIÓN
2. EVOLUCIÓN NATURAL
3. EVOLUCIÓN ARTIFICIAL
4. CONTEXTO
5. APLICACIONES
6. CONCLUSIONES

INTRODUCCIÓN



COMPUTACION EVOLUTIVA

Está compuesta por modelos de evolución basados en poblaciones cuyos elementos representan soluciones a problemas.

La simulación de este proceso en un ordenador resulta ser una técnica de optimización probabilística, que con frecuencia mejora a otros métodos clásicos en problemas difíciles.

Enlace: http://www.aic.nrl.navy.mil/galist/

EVOLUCIÓN NATURAL

En la naturaleza, los procesos evolutivos ocurren cuando se satisfacen las siguientes condiciones:

Una entidad o individuo tiene la habilidad de reproducirse.

Hay una población de tales individuos que son capaces de reproducirse.

Existe alguna variedad, diferencia, entre los individuos que se reproducen.

Algunas diferencias en la habilidad para sobrevivir en el entorno están asociadas con esa variedad.

EVOLUCIÓN NATURAL

Los mecanismos que conducen esta evolución no son totalmente conocidos, pero sí algunas de sus características, que son ampliamente aceptadas:

La evolución es un proceso que opera sobre los cromosomas más que sobre las estructuras de la vida que están codificadas en ellos.

EVOLUCIÓN NATURAL

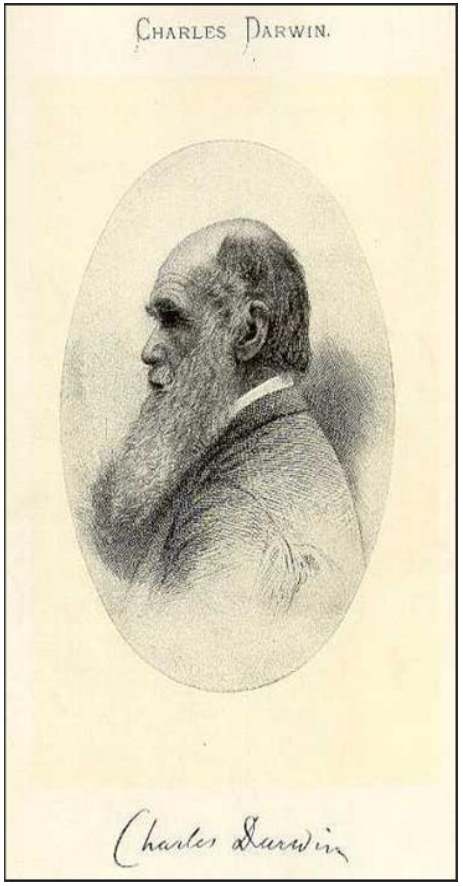
La selección natural es el enlace entre los cromosomas y la actuación de sus estructuras decodificadas.

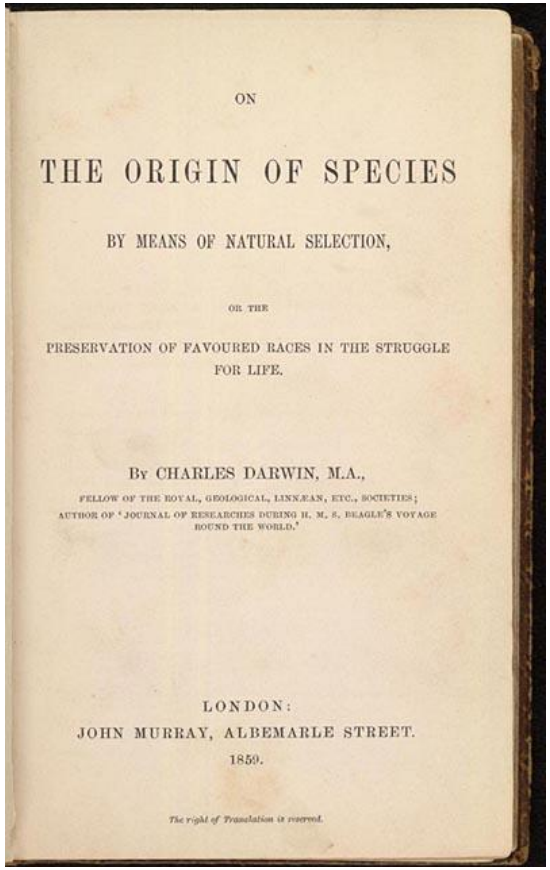
El proceso de reproducción es el punto en el cual la evolución toma parte, actúa.

La evolución biológica no tiene memoria.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection or the Preservations of Favored Races in the Struggle for Life.* London: John Murray.

EVOLUCIÓN NATURAL




ON

THE ORIGIN OF SPECIES

BY MEANS OF NATURAL SELECTION,

OR THE

PRESERVATION OF FAVOURED RACES IN THE STRUGGLE FOR LIFE.

By CHARLES DARWIN, M.A.,

FELLOW OF THE ROYAL, GEOLOGICAL, LINNÆAN, ETC., SOCIETIES;
AUTHOR OF 'JOURNAL OF RESEARCHES DURING H. M. S. BEAGLE'S VOYAGE ROUND THE WORLD.'

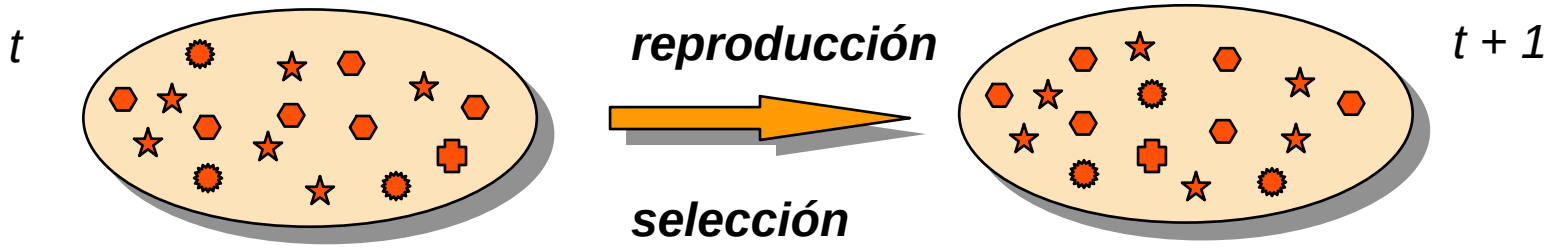
LONDON:
JOHN MURRAY, ALBEMARLE STREET.
1859.

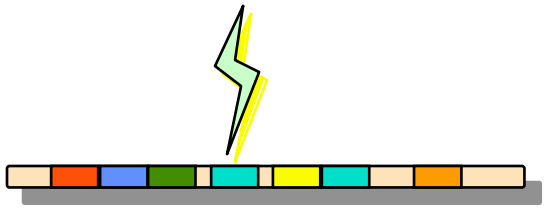
EVOLUCIÓN ARTIFICIAL

LA METÁFORA

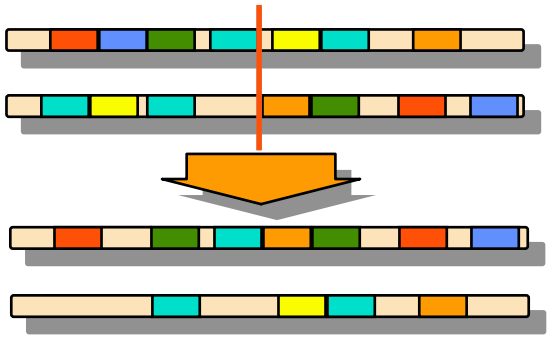
EVOLUCIÓN		RESOLUCIÓN DE PROBLEMAS
Individuo	⟷	Solución Candidata
Adaptación	⟷	Calidad
Entorno	⟷	Problema

EVOLUCIÓN ARTIFICIAL





mutación



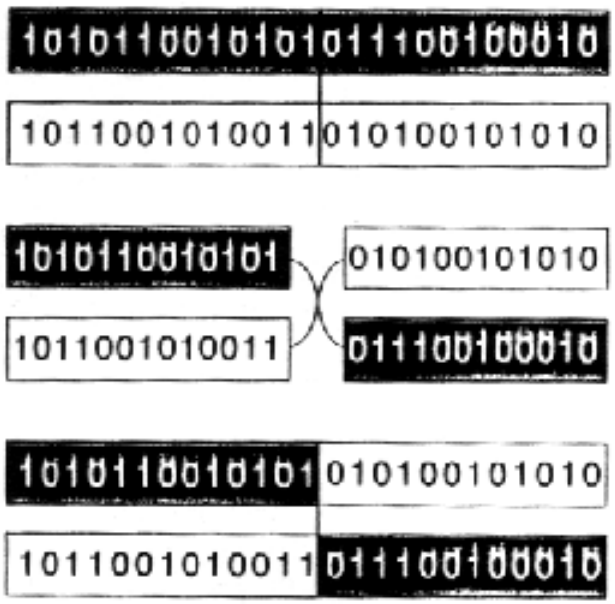
recombinación

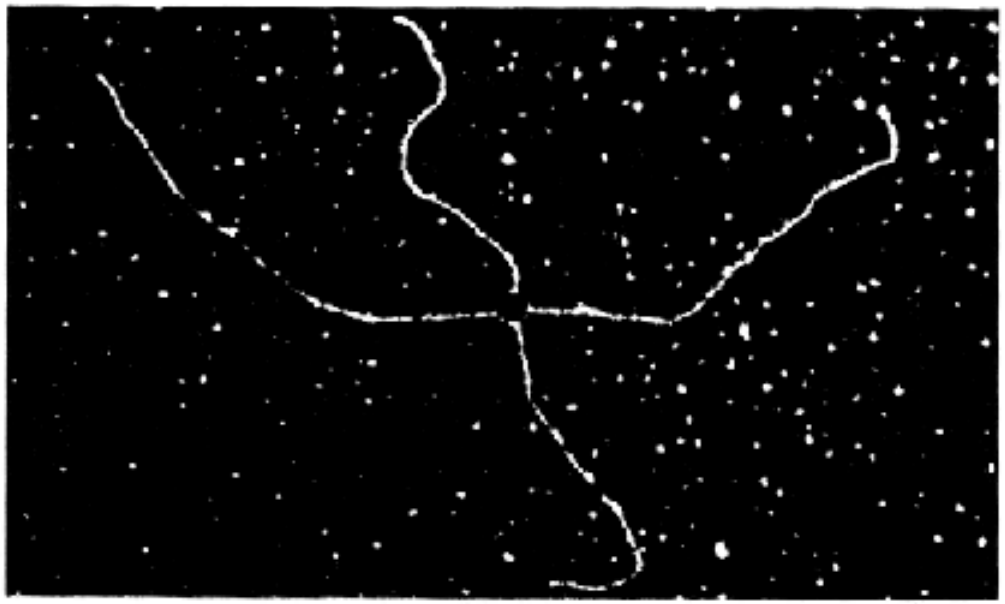
RECOMBINACIÓN OPTATIVA

LOS INGREDIENTES

EVOLUCIÓN ARTIFICIAL

Imagen clásica (John Holland) que introduce el operador de cruce (recombinación)





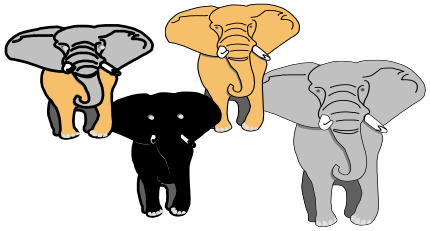
CROSSOVER is the fundamental mechanism of genetic rearrangement for both real organisms and genetic algorithms. Chromosomes line up and then swap the portions of their genetic code beyond the crossover point.

Creditos: Genetic Algorithms, Scientific American, July 1992, 66-72.

EVOLUCIÓN ARTIFICIAL

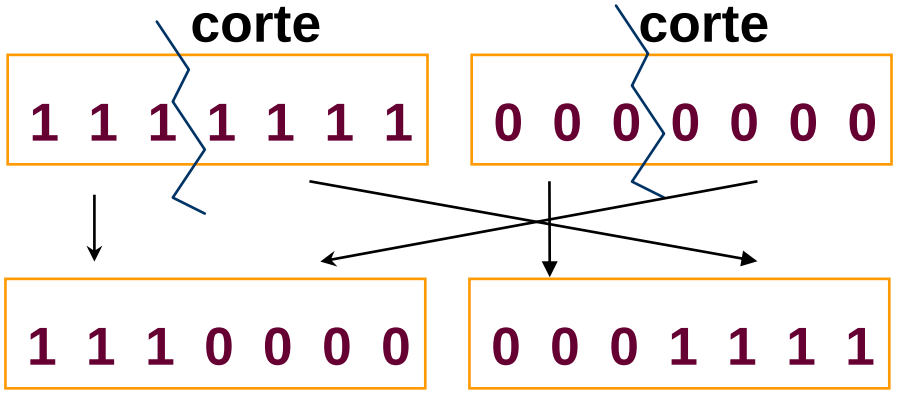
Ejemplo: Recombinación para representación binaria

Población:

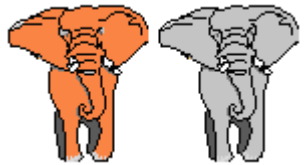


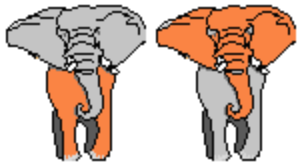
. . .

Cada cromosoma se trocea en n partes las cuales son recombinadas. (Ejemplo para *n=1*)



padres





descendientes

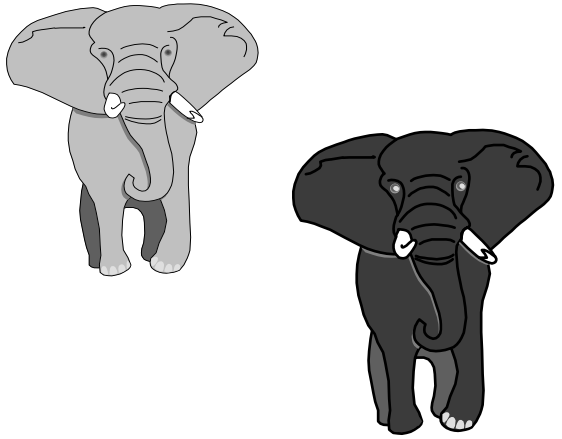
EVOLUCIÓN ARTIFICIAL

Ejemplo: Mutación para representación binaria

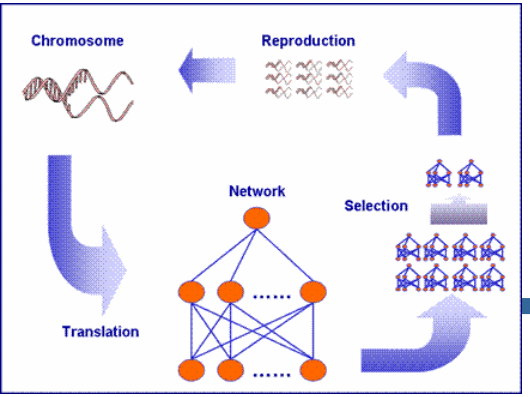
antes **1 1 1 1 1 1 1**

después **1 1 1 0 1 1 1**

gen mutado



La mutación suele ocurrir con probabilidad p_m para cada gen



EVOLUCIÓN ARTIFICIAL

Selección

PADRES

Recombinación

Mutación

DESCENDIENTES

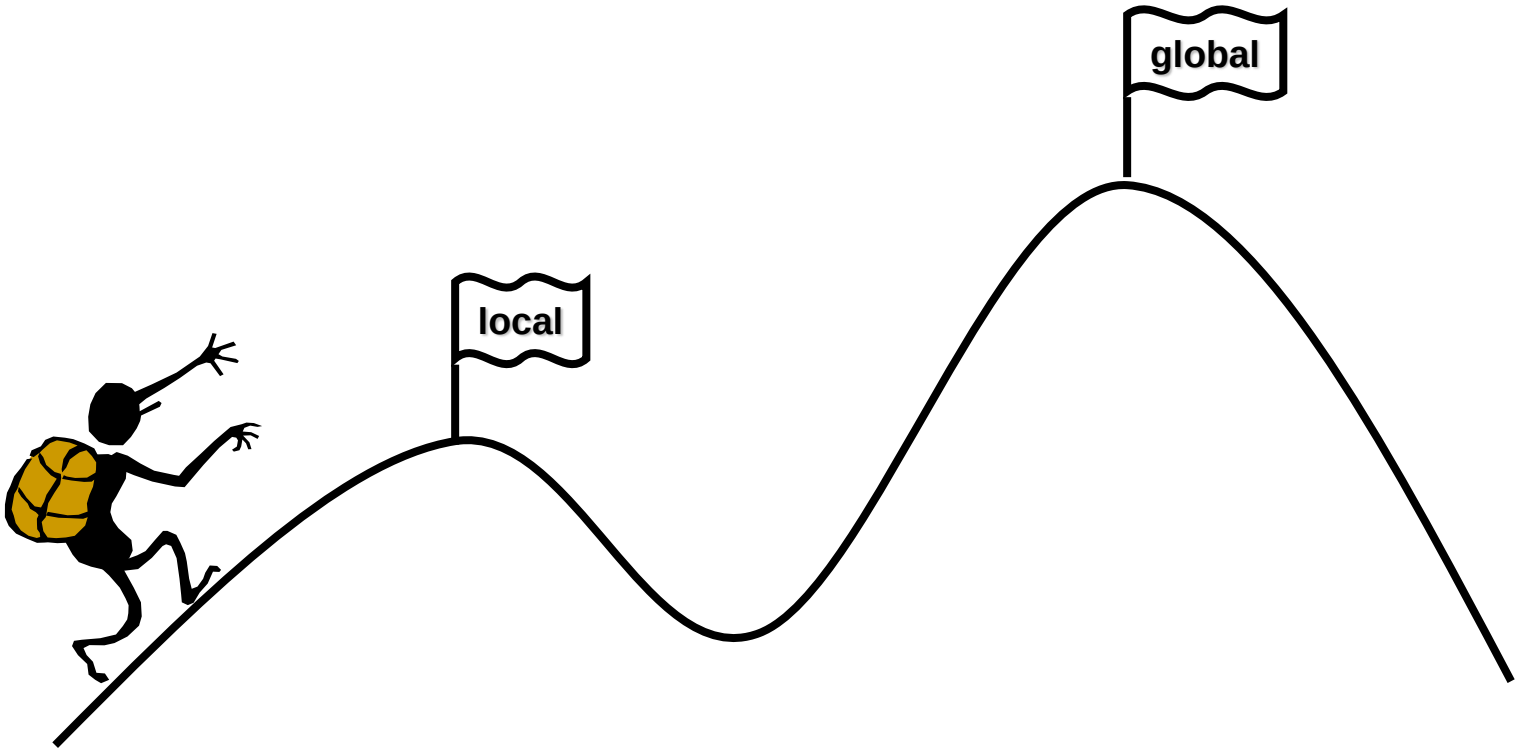
Reemplazamiento

POBLACIÓN

El ciclo de la Evolución

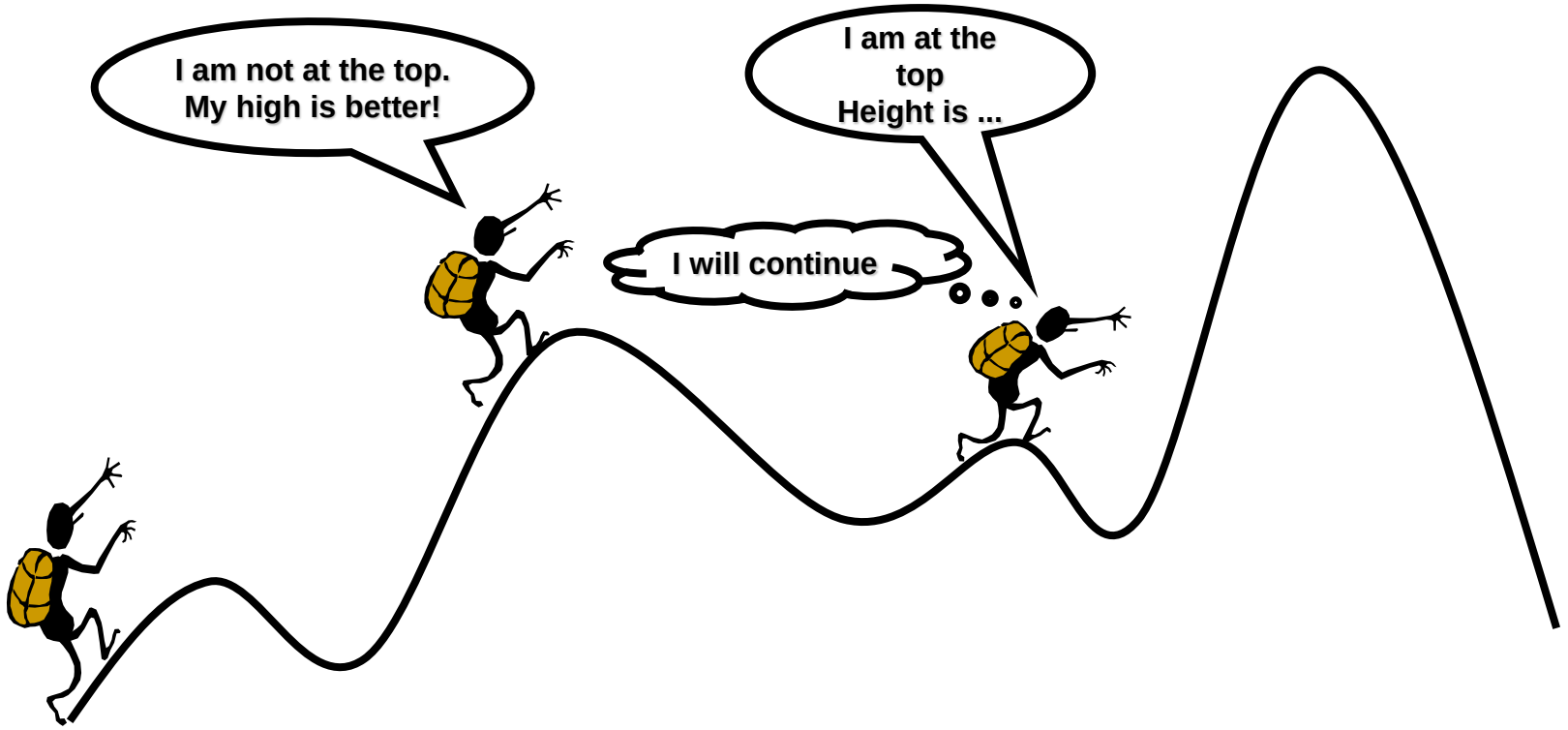
EVOLUCIÓN ARTIFICIAL
(Trayectorias vs poblaciones)

- Búsqueda basada en una trayectoria



EVOLUCIÓN ARTIFICIAL
(Trayectorias vs poblaciones)

- Búsqueda basada en poblaciones



EVOLUCIÓN ARTIFICIAL
(Trayectorias vs poblaciones)

- Búsqueda basada en poblaciones

EVOLUCIÓN ARTIFICIAL

Existen cuatro paradigmas básicos:

John Holland
Inventor of genetic algorithms
Professor of CS and Psychology at the U. of Michigan.

Algoritmos Genéticos que utilizan operadores genéticos sobre cromosomas. **1975, Michigan University**

Inventors of Evolution Strategies

Estrategias de Evolución que enfatizan los cambios de comportamiento al nivel de los individuos. **1964, Technische Universität Berlin**

Hans-Paul Schwefel
Universität Dortmund

Ing. Ingo Rechenberg
Bionics & Evolutiontechnique
Technical University Berlin
http://www.bionik.tu-berlin.de/

Programación Evolutiva que enfatizan los cambios de comportamiento al nivel de las especies. **1960-1966, Florida**

Lawrence J. Fogel,
Natural Selection, Inc.
Inventor of Evolutionary Programming

Programación Genética que evoluciona expresiones representadas como árboles.
1989, Stanford University

John Koza
Stanford University.
Inventor of Genetic Programming

EVOLUCIÓN ARTIFICIAL

Existen otros múltiples Modelos de Evolución de Poblaciones:

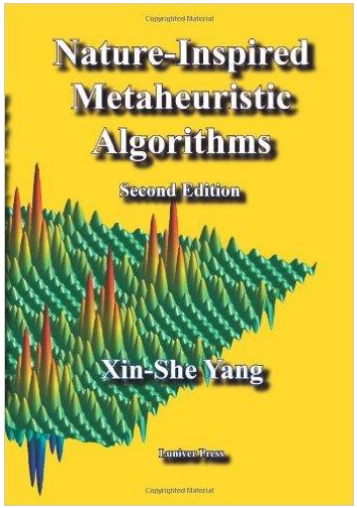
EDA: Estimation Distribution Algorithms (Algoritmos basados en Estimación de Distribuciones)

DE: Differential Evolution (Evolución Diferencial)

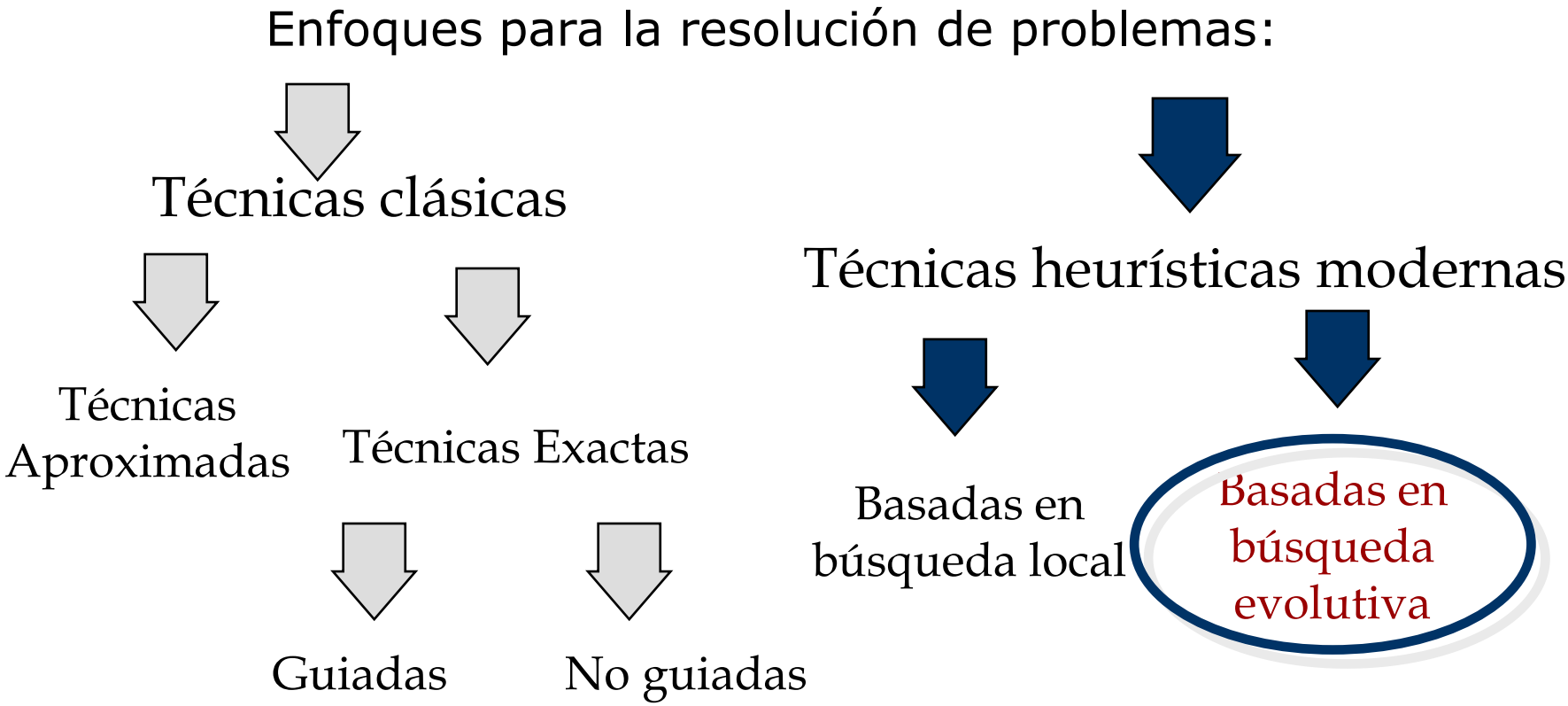
Algoritmos Meméticos

Scatter Search – Búsqueda Dispersa

Firefly Algorithm, Bat algorithm, Cuckoo algorithm, Bee algorithms, …



CONTEXTUALIZACIÓN

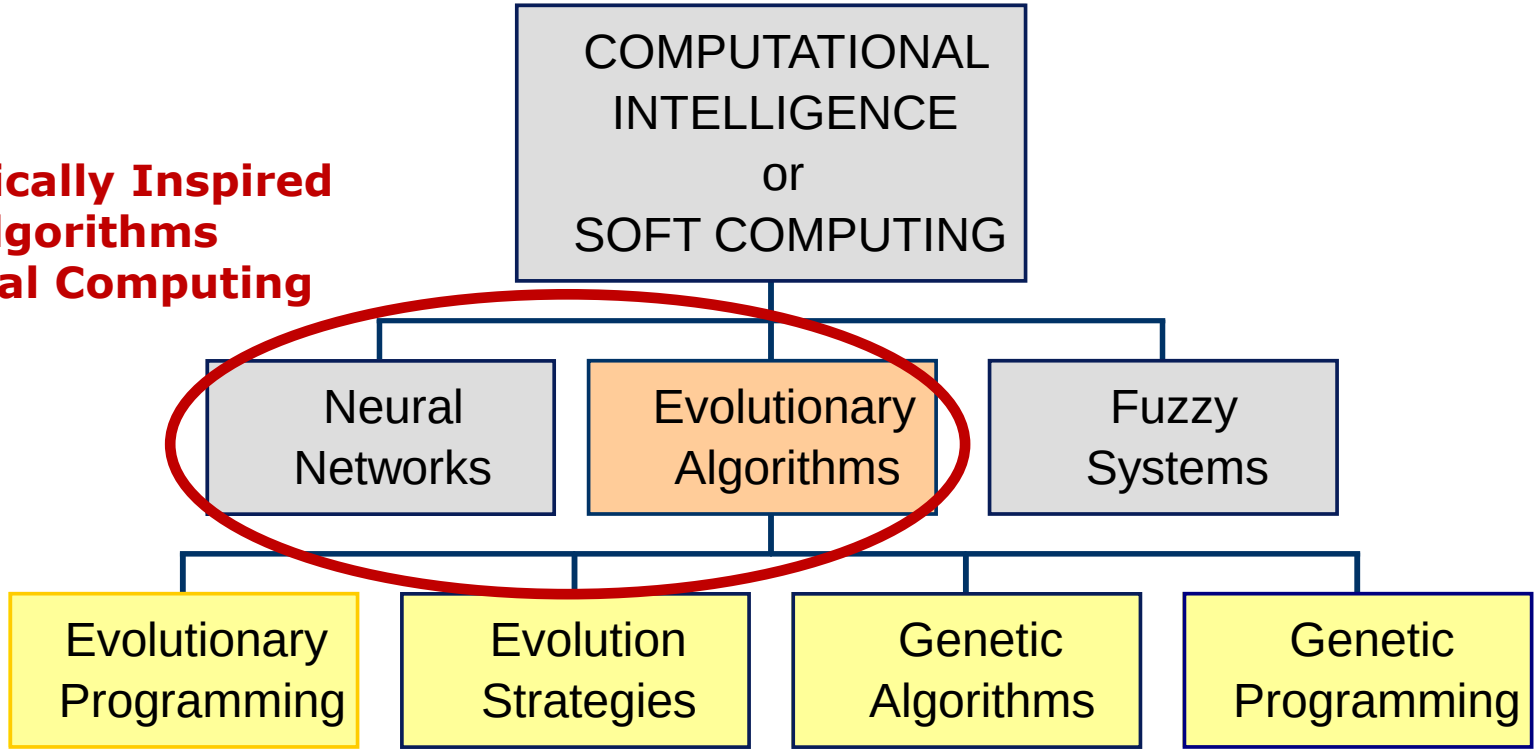


CONTEXTUALIZACIÓN

INTELIGENCIA COMPUTACIONAL

TAXONOMÍA

Biologically Inspired Algorithms
Natural Computing

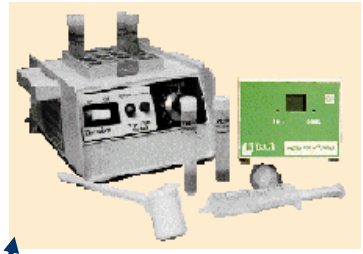


APLICACIONES

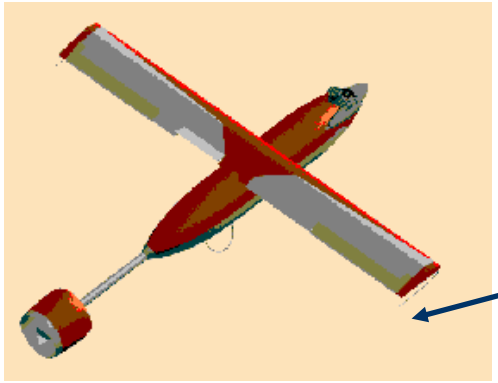
Clasificación

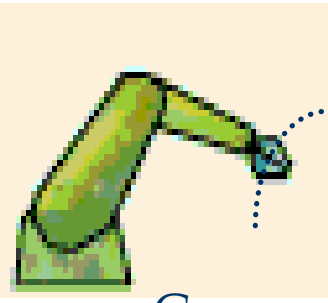
Aprendizaje

Control de procesos químicos



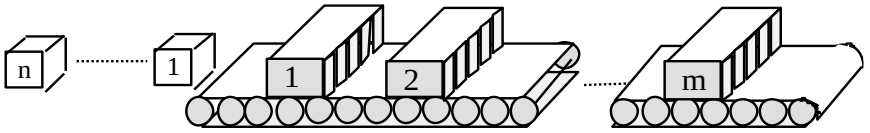
Optimización estructural



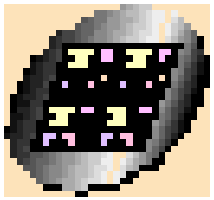


Generación de trayectorias

Planificación de sistemas de Producción



Diseño de circuitos VLSI



APLICACIONES

DOMINIOS DE APLICACIÓN

Optimización combinatoria

Optimización en ingeniería

Modelado e identificación de sistemas

Planificación y control

Aprendizaje y ciencia de datos

Vida artificial

....

CONCLUSIONES

COMPORTAMIENTO

- ✓ **Buena actuación a un costo aceptable en una amplia variedad de problemas**
- ✓ **Paralelismo intrínseco**
- ✓ **Superioridad con respecto a otras técnicas en problemas complejos:**
  - **con muchos parámetros**
  - **relación compleja entre parámetros**
  - **muchos óptimos (locales)**

CONCLUSIONES

VENTAJAS

- Sin restricciones sobre el espacio de soluciones
- Amplia aplicabilidad
- Bajo coste en desarrollo
- Fáciles de hibridar con otras técnicas
- Soluciones interpretables
- Se pueden ejecutar interactivamente
- Proporcionan un conjunto de soluciones

CONCLUSIONES

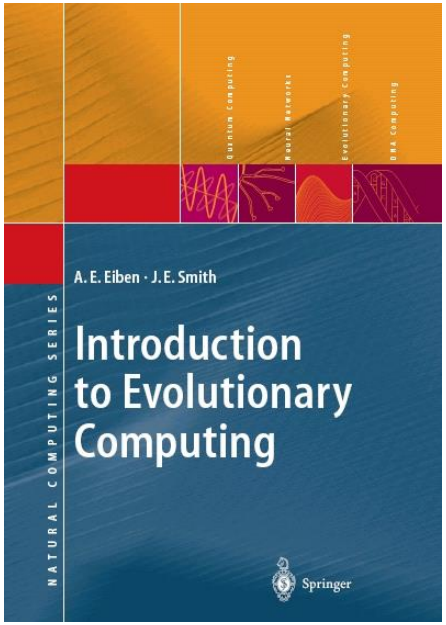
DESVENTAJAS

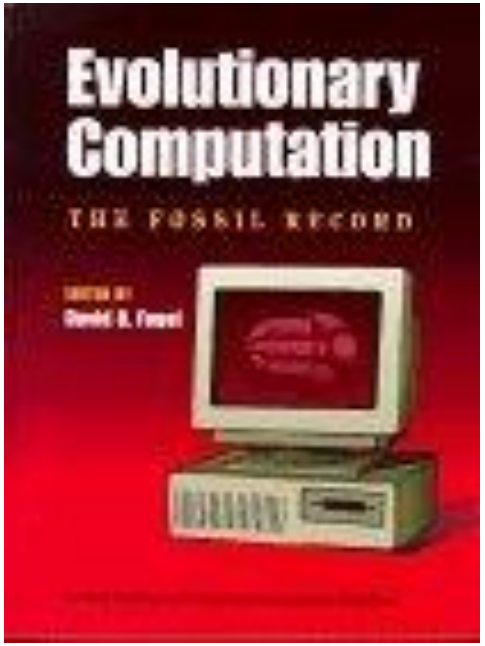
- No garantizan una solución optima en un tiempo finito
- Débil base teórica
- Tienen muchos parámetros a ajustar
- Computacionalmente costosos (lentos)

BIBLIOGRAFÍA

COMPUTACIÓN EVOLUTIVA

A.E. Eiben, J.E. Smith
Introduction to Evolutionary Computation.
Springer Verlag 2003.
(Natural Computing Series)





D.B. Fogel (Ed.)
Evolutionary Computation. The Fossil Record.
(Selected Readings on the
History of Evolutionary Computation).
IEEE Press, 1998.

BIBLIOGRAFÍA

Notes on the Origin of Evolutionary Computation

MOSHE SIPPER

Moshe Sipper is a Senior Researcher in the Logic Systems Laboratory at the Swiss Federal Institute of Technology, Lausanne, Switzerland. He received a B.A. in Computer Science from the Technion—Israel Institute of Technology, and an M.Sc. and a Ph.D. from Tel Aviv University. His chief interests involve the application of biological principles to artificial systems, including evolutionary computation, cellular automata (with an emphasis on evolving cellular machines), bio-inspired systems, evolving hardware, complex adaptive systems, artificial life, and neural networks. Dr. Sipper has authored and coauthored over 60 scientific publications in these areas, including the book Evolution of Parallel Cellular Machines: The Cellular Programming Approach *(Heidelberg: Springer-Verlag, 1997). He was Program Chairman of the Second International Conference on Evolvable Systems: From Biology to Hardware (ICES98), held in Lausanne in September 1998.*

THE ORIGIN OF SPECIES

If during the long course of ages and under varying conditions of life, organic beings vary at all in the several parts of their organisation, and I think this cannot be disputed; if there be, owing to the high geometrical powers of increase of each species, at some age, season, or year, a severe struggle for life, and this certainly cannot be disputed; then, considering the infinite complexity of the relations of all organic beings to each other and to their conditions of existence, causing an infinite diversity in structure, constitution, and habits, to be advantageous to them, I think it would be a most extraordinary fact if no variation ever had occurred useful to each being's own welfare, in the same way as so many variations have occurred useful to man. But if variations useful to any organic being do occur, assuredly individuals thus characterised will have the best chance of being preserved in the struggle for life; and from the strong principle of inheritance they will tend to produce offspring similarly characterised. This principle of preservation, I have called, for the sake of brevity, Natural Selection.

Chapter 4: Natural Selection

In 1859 Charles Darwin published his book *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life.* The shock waves produced by Darwin's revolutionary theory of evolution[1] are still reverberating within society at large, though science has by now come to terms with it.

One of the prominent offshoots of Darwinian evolution occurred almost a century later, with the advent of evolutionary algorithms in the 1950s. The basic idea, by now well entrenched within scientific and engineering lore, is both simple and ingenious: implementing within a computer a simulacrum of evolution by natural selection. The basic principle of evolutionary computation was enunciated by Darwin in his 1859 book: "... one general law, leading to the advancement of all organic beings, namely, multiply, vary, let the strongest live and the weakest die." (Chapter 7)

In the following article, I wish to go back to the origin of evolutionary computation, namely, *The Origin of Species*, with the aim being to expose some ideas that may be relevant to the field today. My strategy is simple: To illustrate each idea, I shall bring forth excerpts from the book, followed by a short annotation. This latter is intended neither to rephrase nor to explain—*The Origin of Species* is remarkable not only for its conceptual depth but also for its clarity of style. Rather, the anno-

Genetic Algorithms

Computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand

by John H. Holland

Living organisms are consummate problem solvers. They exhibit a versatility that puts the best computer programs to shame. This observation is especially galling for computer scientists, who may spend months or years of intellectual effort on an algorithm, whereas organisms come by their abilities through the apparently undirected mechanism of evolution and natural selection.

Pragmatic researchers see evolution's remarkable power as something to be emulated rather than envied. Natural selection eliminates one of the greatest hurdles in software design: specifying in advance all the features of a problem and the actions a program should take to deal with them. By harnessing the mechanisms of evolution, researchers may be able to "breed" programs that solve problems even when no person can fully understand their structure. Indeed, these so-called genetic algorithms have already demonstrated the ability to make breakthroughs in the design of such complex systems as jet engines.

Genetic algorithms make it possible to explore a far greater range of potential solutions to a problem than do conventional programs. Furthermore, as researchers probe the natural selection of programs under controlled and well-understood conditions, the practical results they achieve may yield some insight into the details of how life and intelligence evolved in the natural world.

Most organisms evolve by means of two primary processes: natural selection and sexual reproduction. The first determines which members of a population survive to reproduce, and the second ensures mixing and recombination among the genes of their offspring. When sperm and ova fuse, matching chromosomes line up with one another and then cross over partway along their length, thus swapping genetic material. This mixing allows creatures to evolve much more rapidly than they would if each offspring simply contained a copy of the genes of a single parent, modified occasionally by mutation. (Although unicellular organisms do not engage in mating as humans like to think of it, they do exchange genetic material, and their evolution can be described in analogous terms.)

Selection is simple: if an organism fails some test of fitness, such as recognizing a predator and fleeing, it dies. Similarly, computer scientists have little trouble weeding out poorly performing algorithms. If a program is supposed to sort numbers in ascending order, for example, one need merely check whether each entry of the program's output is larger than the preceding one.

People have employed a combination of crossbreeding and selection for millennia to breed better crops, racehorses or ornamental roses. It is not as easy, however, to translate these procedures for use on computer programs. The chief problem is the construction of a "genetic code" that can represent the structure of different programs, just as DNA represents the structure of a person or a mouse. Mating or mutating the text of a FORTRAN program, for example, would in most cases not produce a better or worse FORTRAN program but rather no program at all.

The first attempts to mesh computer science and evolution, in the late 1950s and early 1960s, fared poorly because they followed the emphasis in biological texts of the time and relied on mutation rather than mating to generate new gene combinations. Then, in the early 1960s, Hans J. Bremermann of the University of California at Berkeley added a kind of mating: the characteristics of offspring were determined by summing up corresponding genes in the two parents. This mating procedure was limited, however, because it could apply only to characteristics that could be added together in a meaningful way.

During this time, I had been investigating mathematical analyses of adaptation and had become convinced that the recombination of groups of genes by means of mating was a critical part of evolution. By the mid-1960s I had developed a programming technique, the genetic algorithm, that is well suited to evolution by both mating and mutation. During the next decade, I worked to extend the scope of genetic algorithms by creating a genetic code that could represent the structure of any computer program.

The result was the classifier system, consisting of a set of RULES, each of which performs particular actions every time its conditions are satisfied by some piece of information. The conditions and actions are represented by strings of bits corresponding to the presence or absence of specific characteristics in the rules' input and output. For each characteristic that was present, the string would contain a 1 in the appropriate position, and for each that was absent, it would contain a 0. For example, a classifier rule that recognized dogs might be encoded as a string containing 1's for the bits corresponding to "hairy," "slobbers," "barks," "loyal" and "chases sticks" and 0's for the bits corresponding to "metallic," "speaks Urdu" and "possesses credit cards." More realistically, the programmer should choose the simplest, most primitive characteristics so that they can be combined—as

JOHN H. HOLLAND has been investigating the theory and practice of algorithmic evolution for nearly 40 years. He is a professor of psychology and of electrical engineering and computer science at the University of Michigan. Holland received a B.S. in physics from the Massachusetts Institute of Technology in 1950 and served on the Logical Planning Group for IBM's first programmed electronic computer (the 701) from 1950 until 1952. He received an M.A. in mathematics and a Ph.D. in communication sciences from the University of Michigan. Holland has been a member of the Steering Committee of the Santa Fe Institute since its inception in 1987 and is an external professor there.

METAHEURÍSTICAS

TEMA 3. METAHEURÍSTICAS BASADAS EN POBLACIONES

Parte I:

1. **INTRODUCCIÓN A LA COMPUTACIÓN EVOLUTIVA**
2. **ALGORITMOS GENÉTICOS**
3. **ALGORITMOS GENÉTICOS. EXPLORACIÓN VS EXPLOTACIÓN**
4. **CONCLUSIONES**

BIBLIOGRAFÍA
D.B. Fogel (Ed.). Evolutionary Computation. The Fossil Record. (Selected Readings on the History of Evolutionary Computation). IEEE Press, 1998.
A.E. Eiben, J.E. Smith. Introduction to Evolutionary Computation. Springer Verlag 2003.

METAHEURÍSTICAS

TEMA 3. METAHEURÍSTICAS BASADAS EN POBLACIONES

ALGORITMOS GENÉTICOS

1. INTRODUCCIÓN A LOS ALGORITMOS GENÉTICOS
2. MODELOS: GENERACIONAL vs ESTACIONARIO
3. ¿CÓMO SE CONSTRUYE UN AG?
4. SOBRE SU UTILIZACIÓN
5. EJEMPLO: VIAJANTE DE COMERCIO
6. CONCLUSIONES

1. INTRODUCCIÓN A LOS ALGORITMOS GENÉTICOS

- ¿QUÉ ES UN ALGORITMO GENÉTICO?
- LOS INGREDIENTES
- El CICLO DE LA EVOLUCIÓN
- ESTRUCTURA DE UN ALGORITMO GENÉTICO

¿Qué es un Algoritmo Genético?

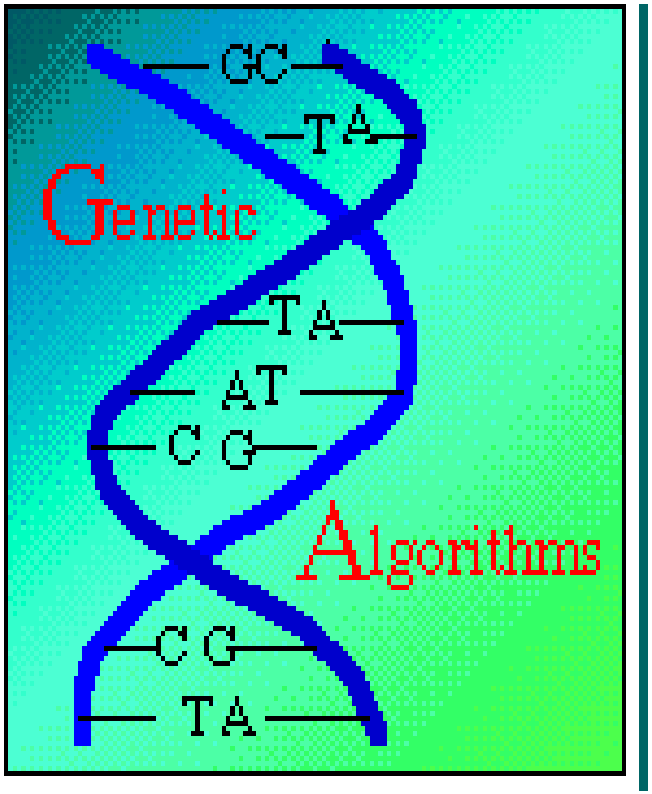
Los Algoritmos Genéticos

son algoritmos de optimización
búsqueda
y aprendizaje
inspirados en los procesos de

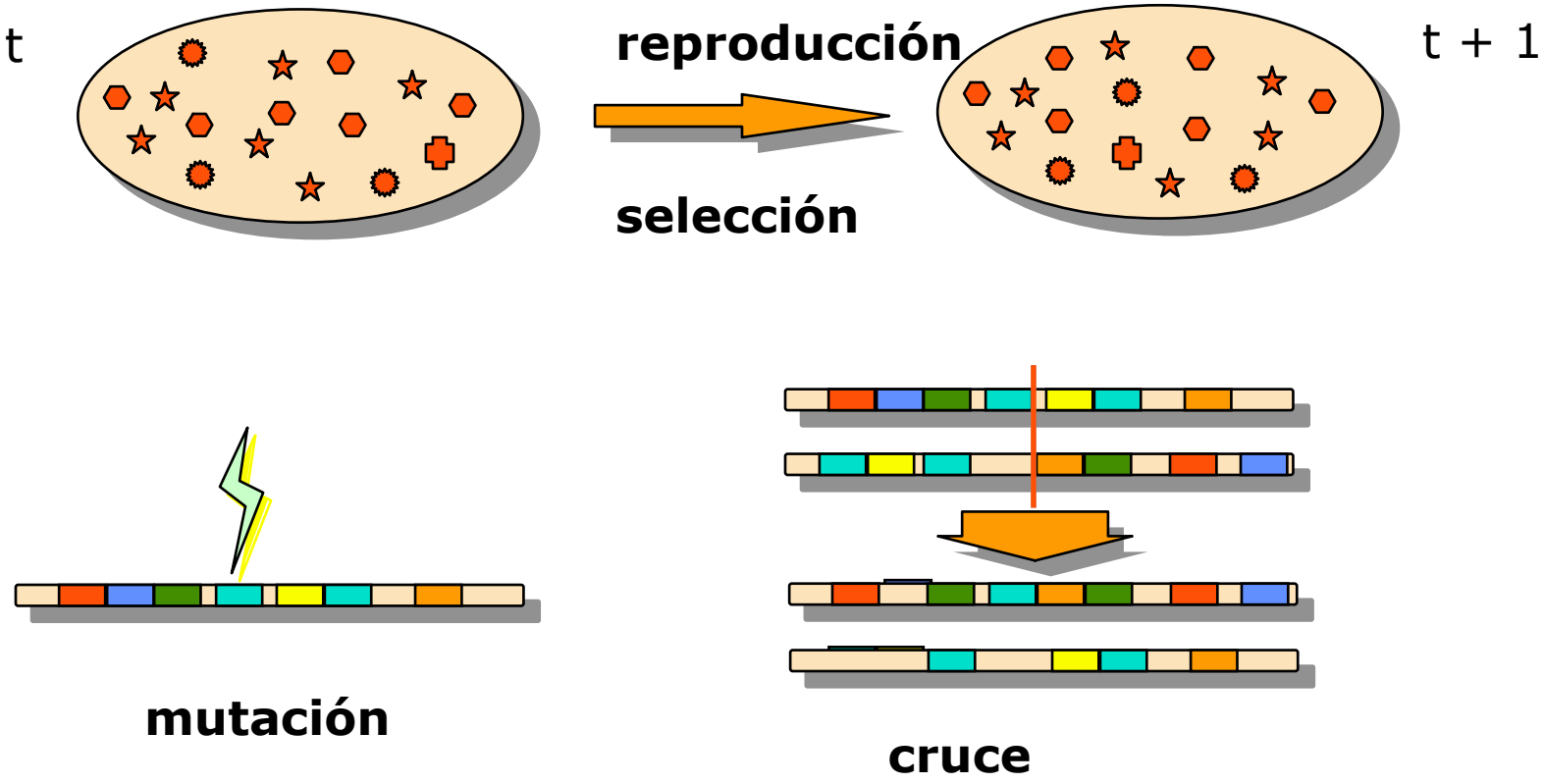
Evolución Natural

y

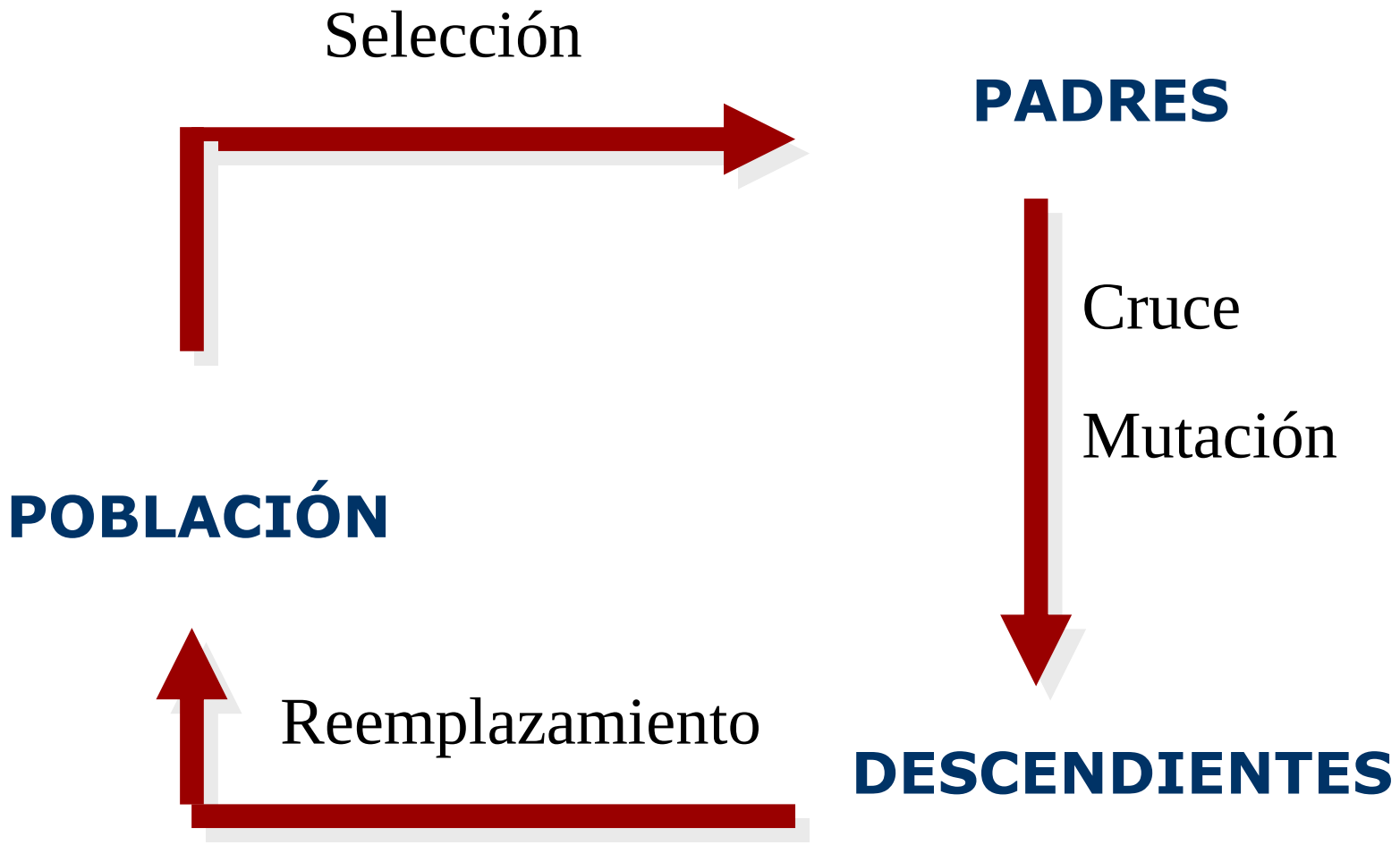
Evolución Genética



Los Ingredientes



El ciclo de la Evolución



Estructura de un Algoritmo Genético

Procedimiento Algoritmo Genético

```
Inicio (1)
    t = 0;
    inicializar P(t);
    evaluar P(t);
    Mientras (no se cumpla la condición de parada) hacer
    Inicio(2)
        t = t + 1
        seleccionar P' desde P(t-1)
        recombinar P'
        mutar P'
        reemplazar P(t) a partir de P(t-1) y P'
        evaluar P(t)
    Final(2)
Final(1)
```

2. MODELOS: GENERACIONAL vs ESTACIONARIO

Modelo generacional. Durante cada iteración se crea una población completa con nuevos individuos.

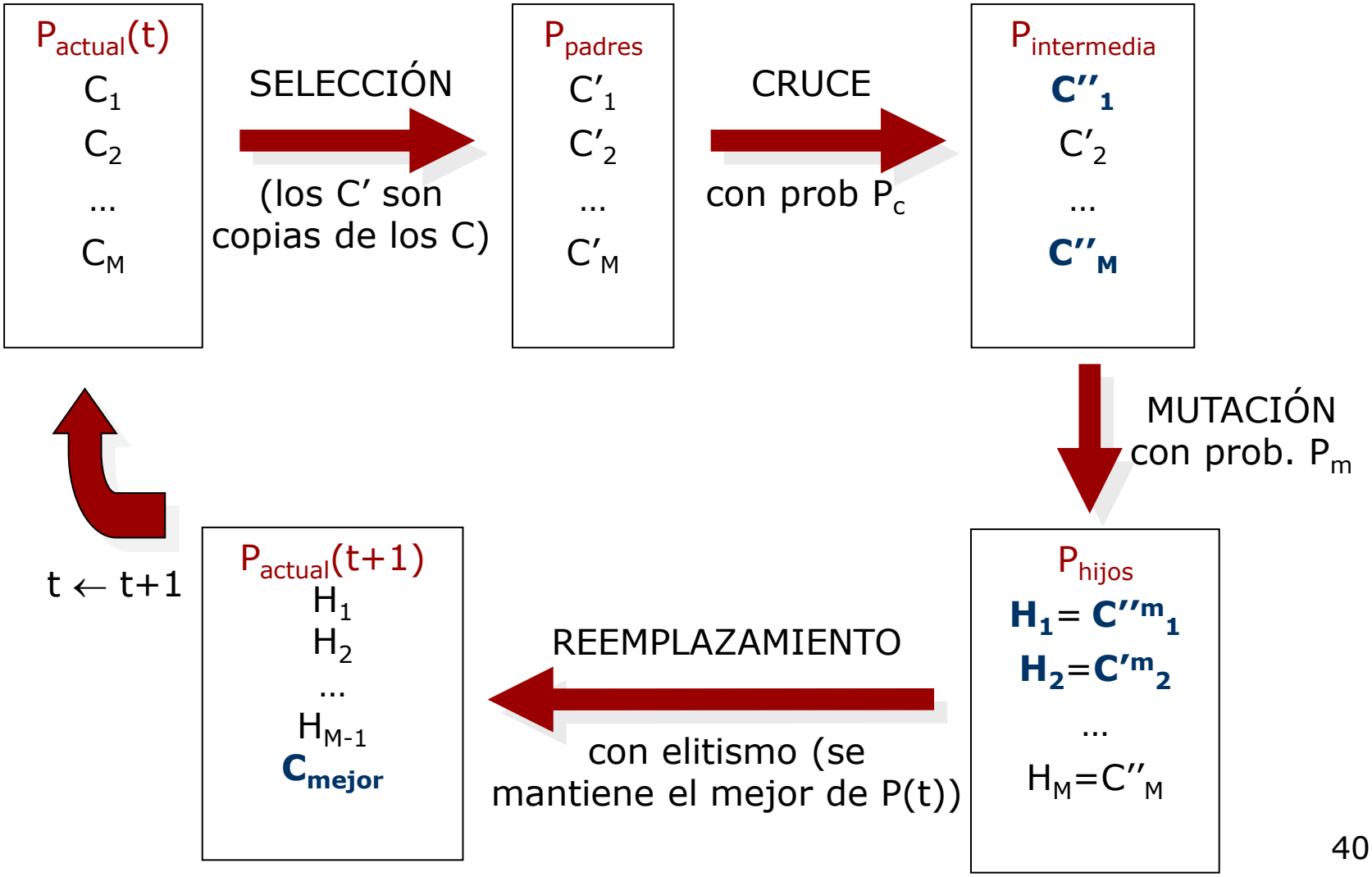
La nueva población reemplaza directamente a la antigua.

Modelo estacionario. Durante cada iteración se escogen dos padres de la población (diferentes mecanismos de muestreo) y se les aplican los operadores genéticos.

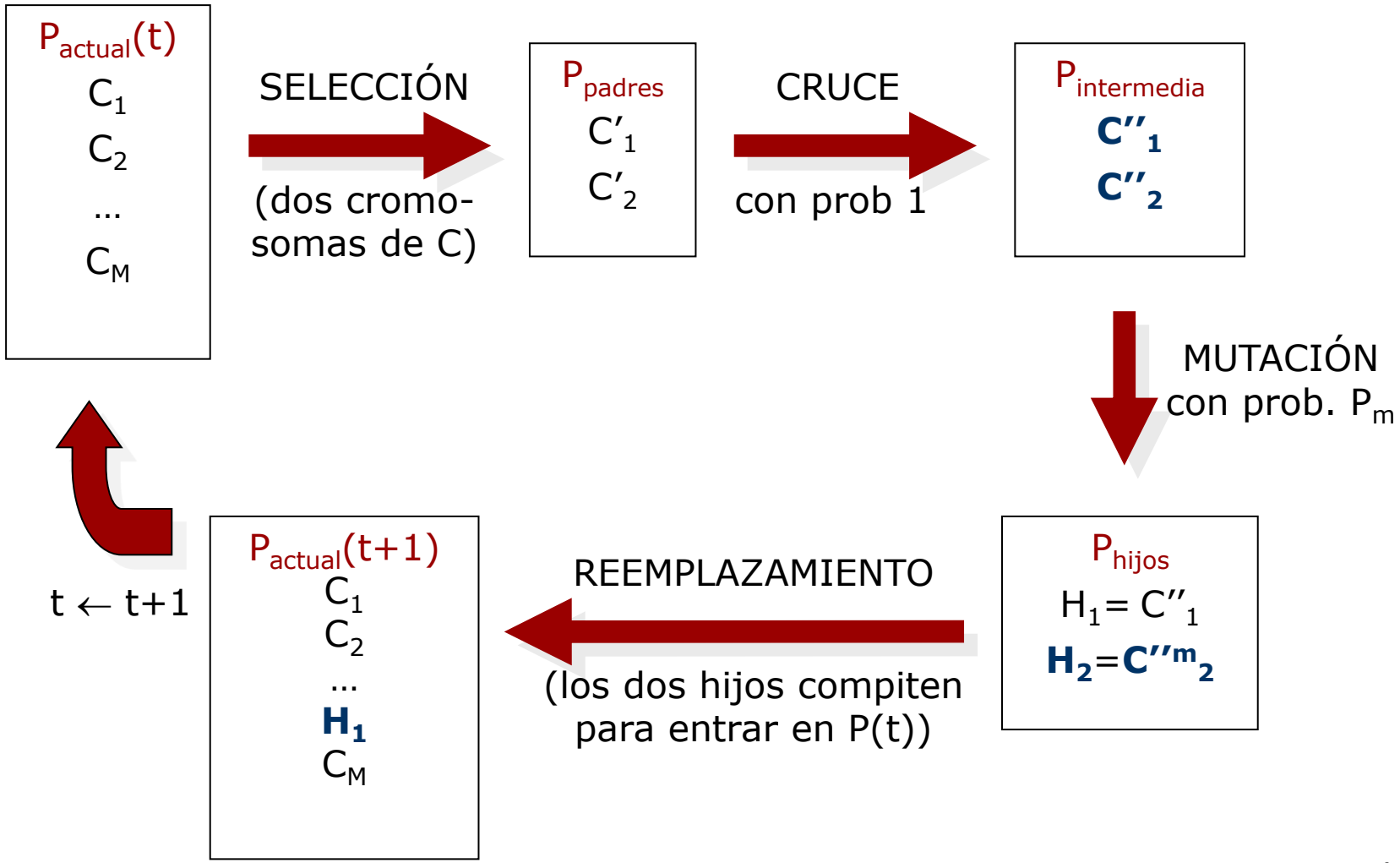
El/los descendiente/s reemplazan a uno/dos cromosoma/s de la población anterior.

El modelo estacionario es elitista. Además produce una presión selectiva alta (convergencia rápida) cuando se reemplazan los peores cromosomas de la población.

Modelo Generacional



Modelo Estacionario



3. ¿CÓMO SE CONSTRUYE UN AG?

Los pasos para construir un Algoritmo Genético

DEPENDE DEL PROBLEMA

- Diseñar una representación
- Decidir cómo inicializar una población
- Diseñar una correspondencia entre genotipo y fenotipo
- Diseñar una forma de evaluar un individuo

COMPONENTES DEL ALGORITMO

- Diseñar un operador de mutación adecuado
- Diseñar un operador de cruce adecuado
- Decidir cómo seleccionar los individuos para ser padres
- Decidir cómo reemplazar a los individuos
- Decidir la condición de parada

Representación

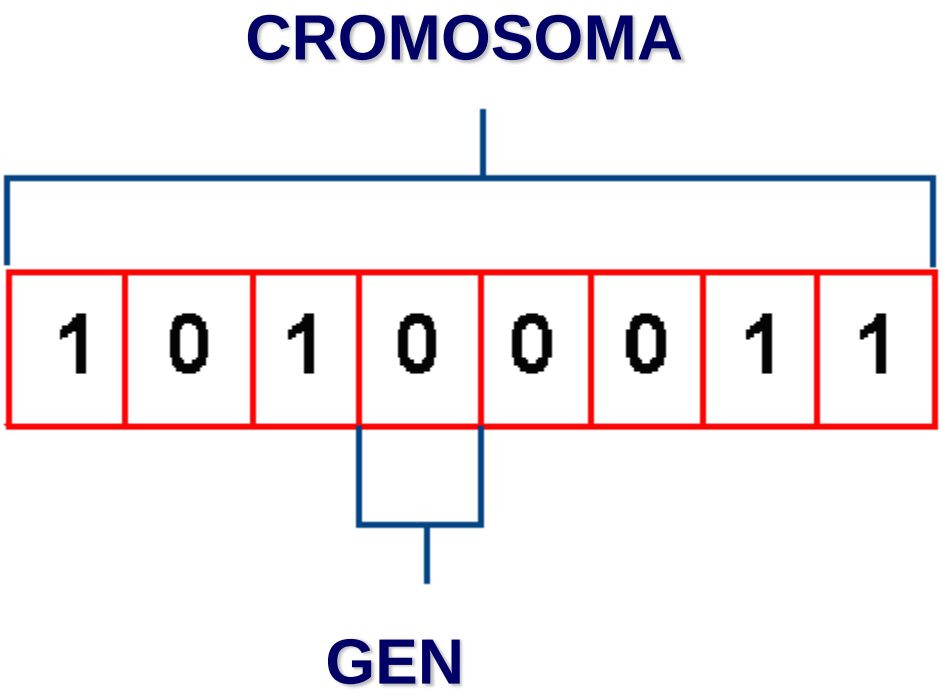
Debemos disponer de un mecanismo para codificar un individuo como un genotipo.

Existen muchas maneras de hacer esto y se ha de elegir la más relevante para el problema en cuestión.

Una vez elegida una representación, hemos de tener en mente como los genotipos (codificación) serán evaluados y qué operadores genéticos hay que utilizar.

Ejemplo: Representación binaria

- La representación de un individuo se puede hacer mediante una codificación discreta, y en particular binaria.



Ejemplo: Representación binaria

8 bits Genotipo

Fenotipo

- **Entero**
- **Número real**
- **secuencia**
- ...
- **Cualquier otra?**

Ejemplo: Representación Real

- Una forma natural de codificar una solución es utilizando valores reales como genes

- Muchas aplicaciones tienen esta forma natural de codificación

Ejemplo: Representación Real

- Los individuos se representan como vectores de valores reales:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, x_i \in R$$

- La función de evaluación asocia a un vector un valor real de evaluación:

$$f : R^n \rightarrow R$$

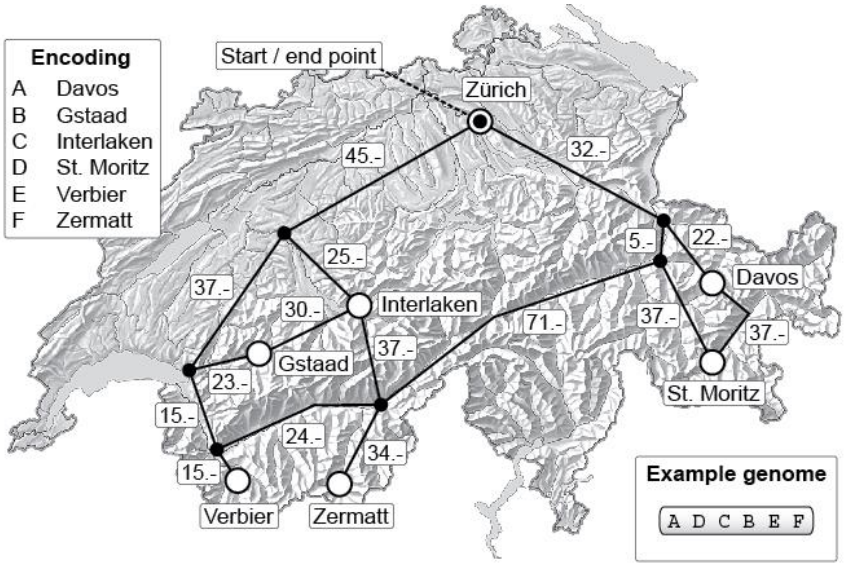
Ejemplo: Representación de orden

- Los individuos se representan como permutaciones.

7	3	6	8	2	4	1	5

- Se utilizan para problemas de secuenciación.

Ejemplo: Viajante de Comercio, donde cada ciudad tiene asignado un único número entre 1 y n. Necesita operadores especiales para garantizar que el resultado de aplicar un operador sigue siendo una permutación.

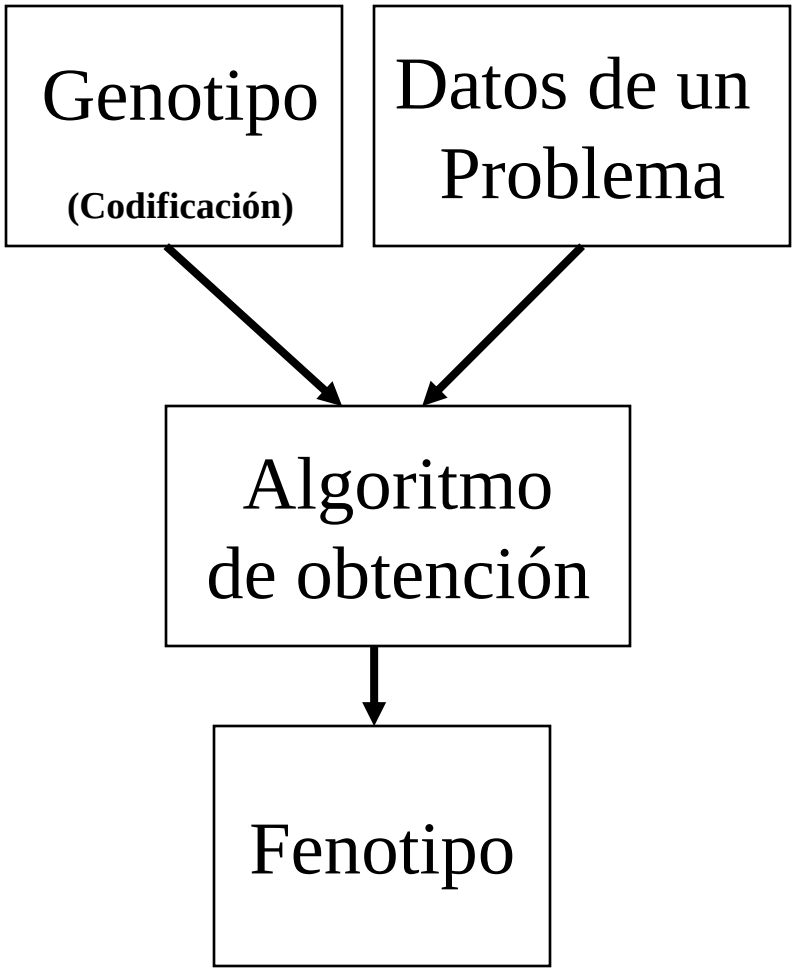


Inicialización

- Uniforme sobre el espacio de búsqueda … (si es posible)
  - Cadena binaria: 0 ó 1 con probabilidad 0.5
  - Representación real: uniforme sobre un intervalo dado (para valores acotados)
- Elegir la población a partir de los resultados de una heurística previa.

Correspondencia entre Genotipo y Fenotipo

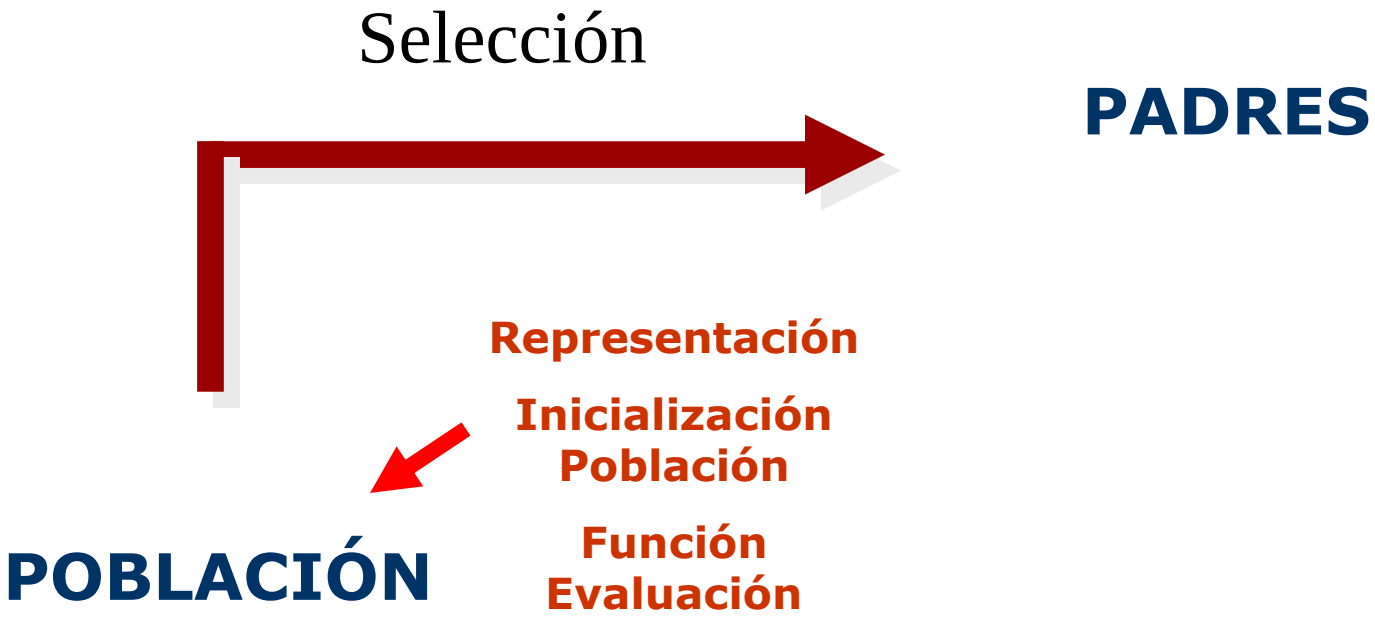
- Algunas veces la obtención del fenotipo a partir del genotipo es un proceso obvio.
- En otras ocasiones el genotipo puede ser un conjunto de parámetros para algún algoritmo, el cual trabaja sobre los datos de un problema para obtener un fenotipo



Evaluación de un individuo

- Este es el paso más costoso para una aplicación real
- Puede ser una subrutina, un simulador, o cualquier proceso externo (ej. Experimentos en un robot, ...)
- Se pueden utilizar funciones aproximadas para reducir el costo de evaluación.
- Cuando hay restricciones, éstas se pueden introducir en el costo como penalización.
- Con múltiples objetivos se busca una solución de compromiso.

¿CÓMO SE CONSTRUYE UN AG?



Estrategia de Selección

Debemos de garantizar que los mejores individuos tienen una mayor posibilidad de ser padres (reproducirse) frente a los individuos menos buenos.

Debemos de ser cuidadosos para dar una oportunidad de reproducirse a los individuos menos buenos. Éstos pueden incluir material genético útil en el proceso de reproducción.

Esta idea nos define la **presión selectiva** que determina en qué grado la reproducción está dirigida por los mejores individuos.

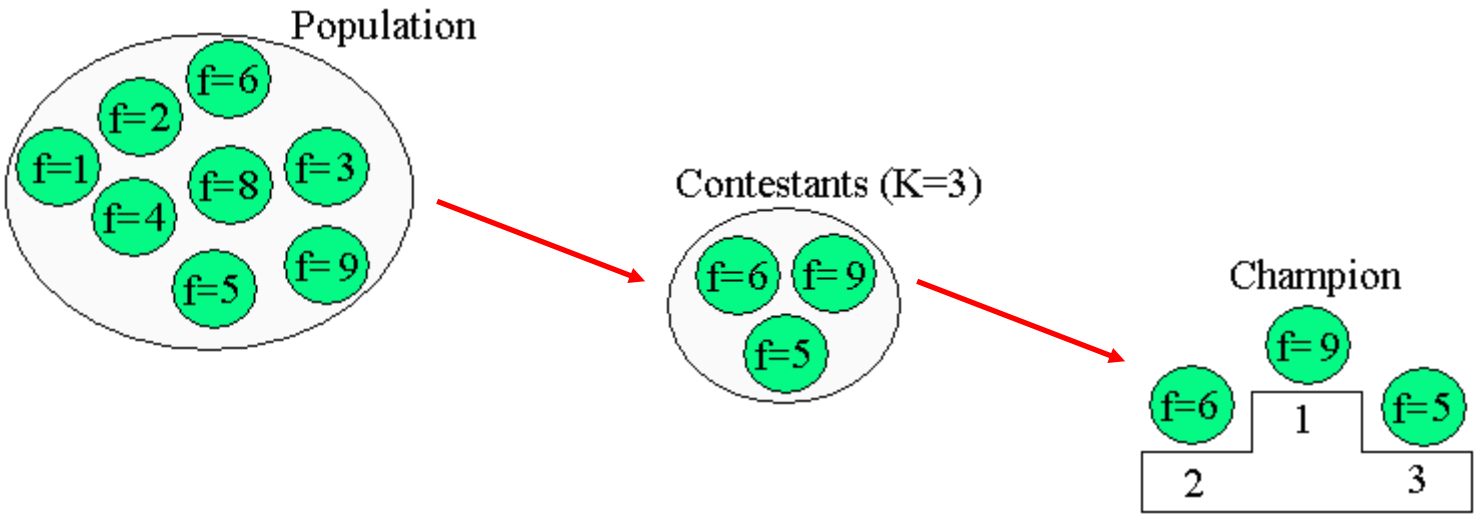
Estrategia de Selección

Selección por torneo

Para cada padre a seleccionar:

- Escoger aleatoriamente k individuos, con reemplazamiento
- Seleccionar el mejor de ellos

k se denomina tamaño del torneo. A mayor k, mayor presión selectiva y viceversa.

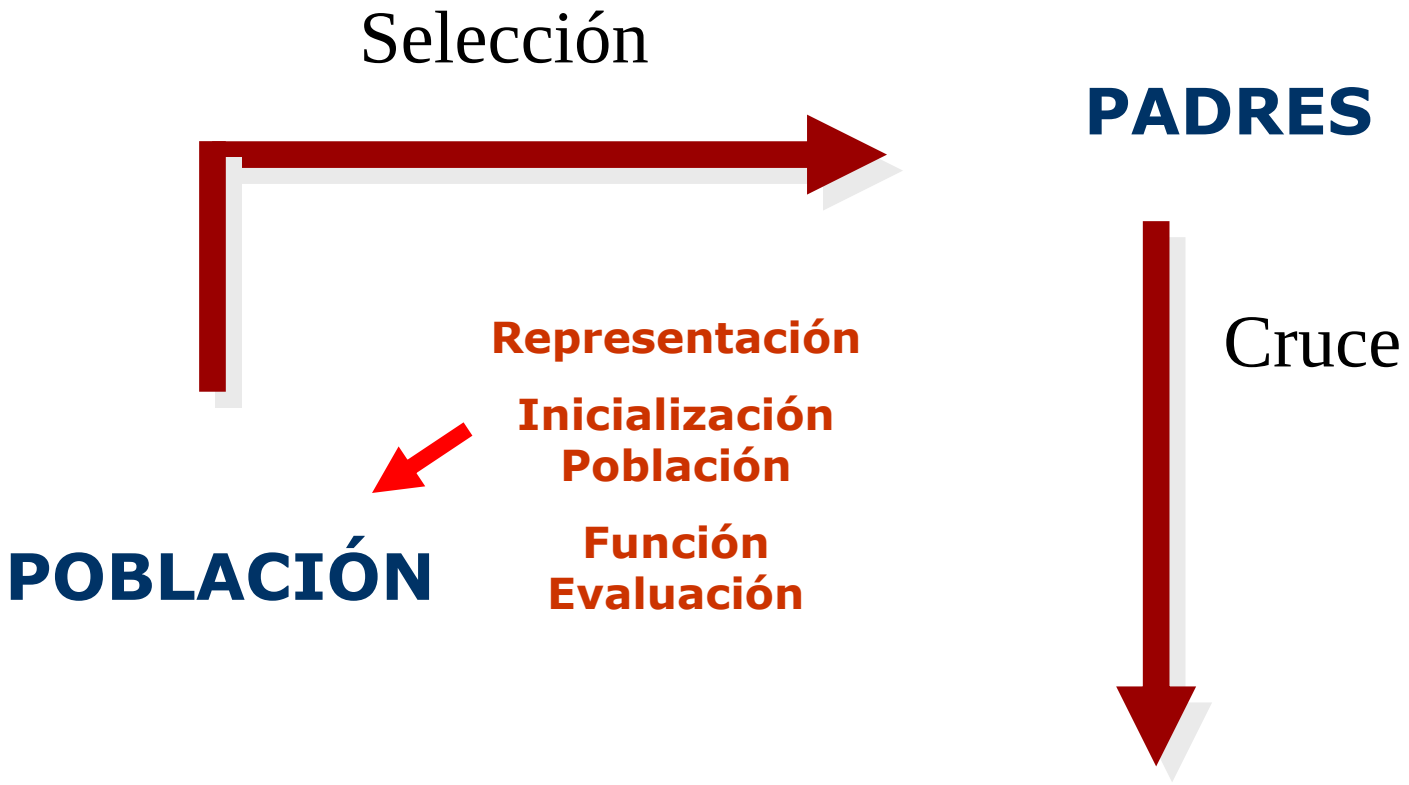


Estrategia de Selección

Algunos esquemas de selección

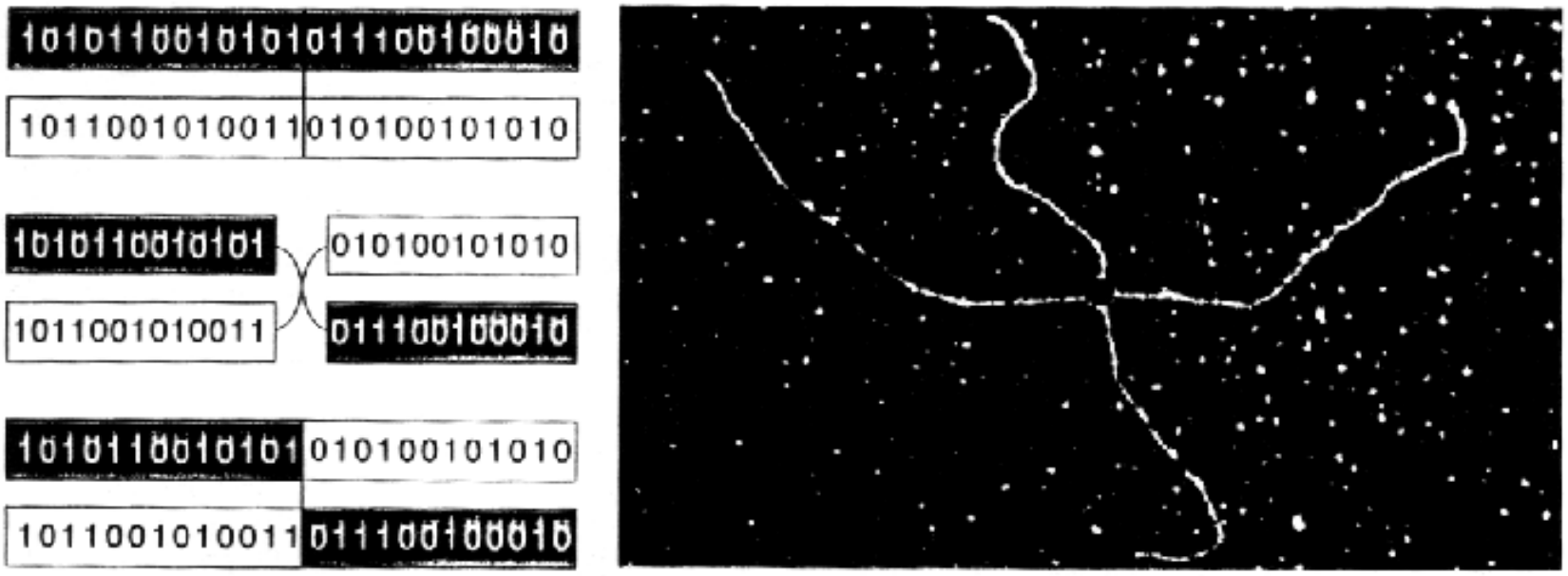
- Selección por Torneo (TS): Escoge al individuo de mejor fitness de entre Nts individuos seleccionados aleatoriamente (Nts=2,3,…).
- Orden Lineal (LR): La población se ordena en función de su fitness y se asocia una probabilidad de selección a cada individuo que depende de su orden.
- Selección Aleatoria (RS).
- Emparejamiento Variado Inverso (NAM): Un padre lo escoge aleatoriamente, para el otro selecciona Nnam padres y escoge el más lejano al primer (Nnam=3,5, ….). Está orientado a generar diversidad.
- Selección por Ruleta: Se asigna una probabilidad de selección proporcional al valor del fitness del cromosoma. (Modelo clásico)

¿CÓMO SE CONSTRUYE UN AG?



Operador de Cruce

Imagen clásica (John Holland) que introduce el operador de cruce



CROSSOVER is the fundamental mechanism of genetic rearrangement for both real organisms and genetic algorithms. Chromosomes line up and then swap the portions of their genetic code beyond the crossover point.

Creditos: Genetic Algorithms, Scientific American, July 1992, 66-72.

Operador de Cruce

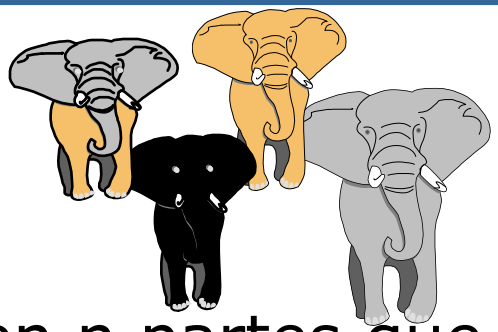
Podríamos tener uno o más operadores de cruce para nuestra representación.

Algunos aspectos importantes a tener en cuenta son:

- Los hijos deberían heredar algunas características de **cada** padre. Si éste no es el caso, entonces estamos ante un operador de mutación.
- Se debe diseñar de acuerdo a la representación.
- La recombinación debe producir cromosomas válidos.
- Se utiliza con una probabilidad alta de actuación sobre cada pareja de padres a cruzar (Pc entre 0.6 y 0.9), si no actua los padres son los descendientes del proceso de recombinación de la pareja.

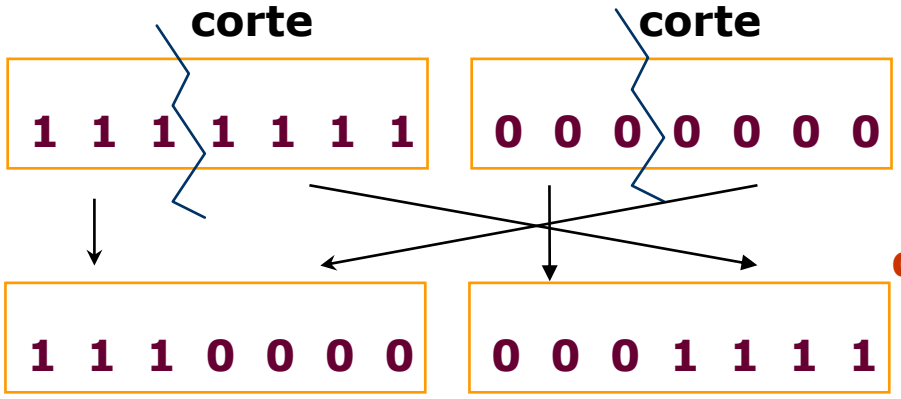
Ejemplo: Operador de cruce para representación binaria

Población:

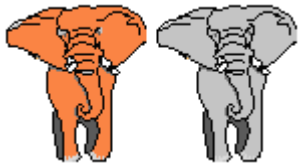


. . .

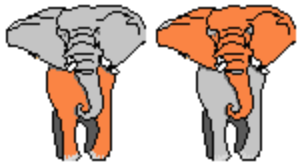
Cada cromosoma se corta en n partes que son recombinadas. (Ejemplo para n = 1).



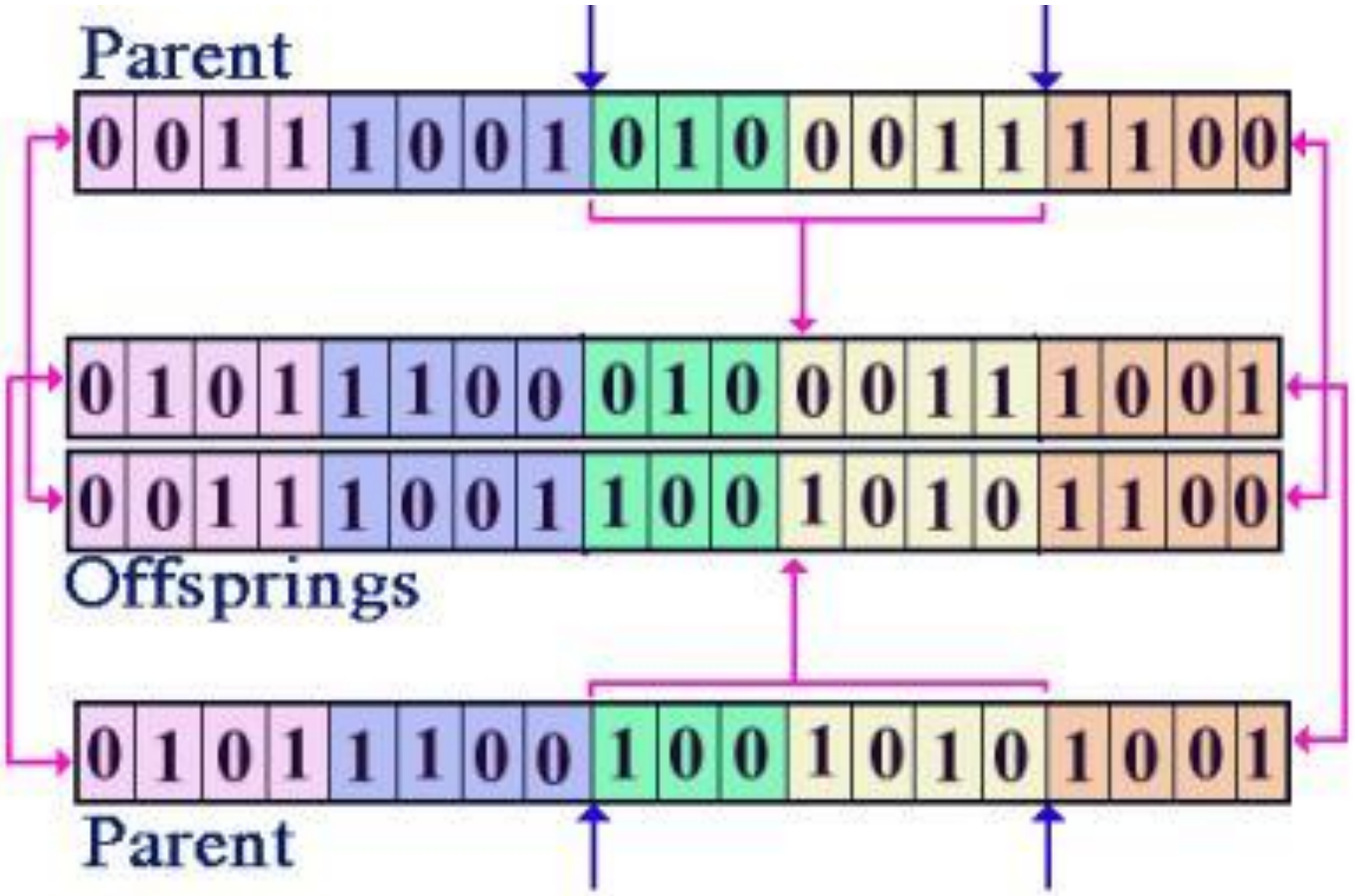
padres



descendientes



Ejemplo: Operador de cruce en dos puntos para representación binaria



Ejemplo: Operador de cruce para representación real

Recombinación aritmética (cruce aritmético):

a	b	c	d	e	f

A	B	C	D	E	F

(a+A)/2	(b+B)/2	(c+C)/2	(d+D)/2	(e+E)/2	(f+F)/2

Existe muchos operadores específicos para la codificación real.

Ejemplo: Operador de cruce para representación real: **BLX-α**

- Dados 2 cromosomas

$$C_1 = (c_{11},\ldots, c_{1n}) \text{ y } C_2 = (c_{21},\ldots, c_{2n}) ,$$

- BLX- α genera dos descendientes

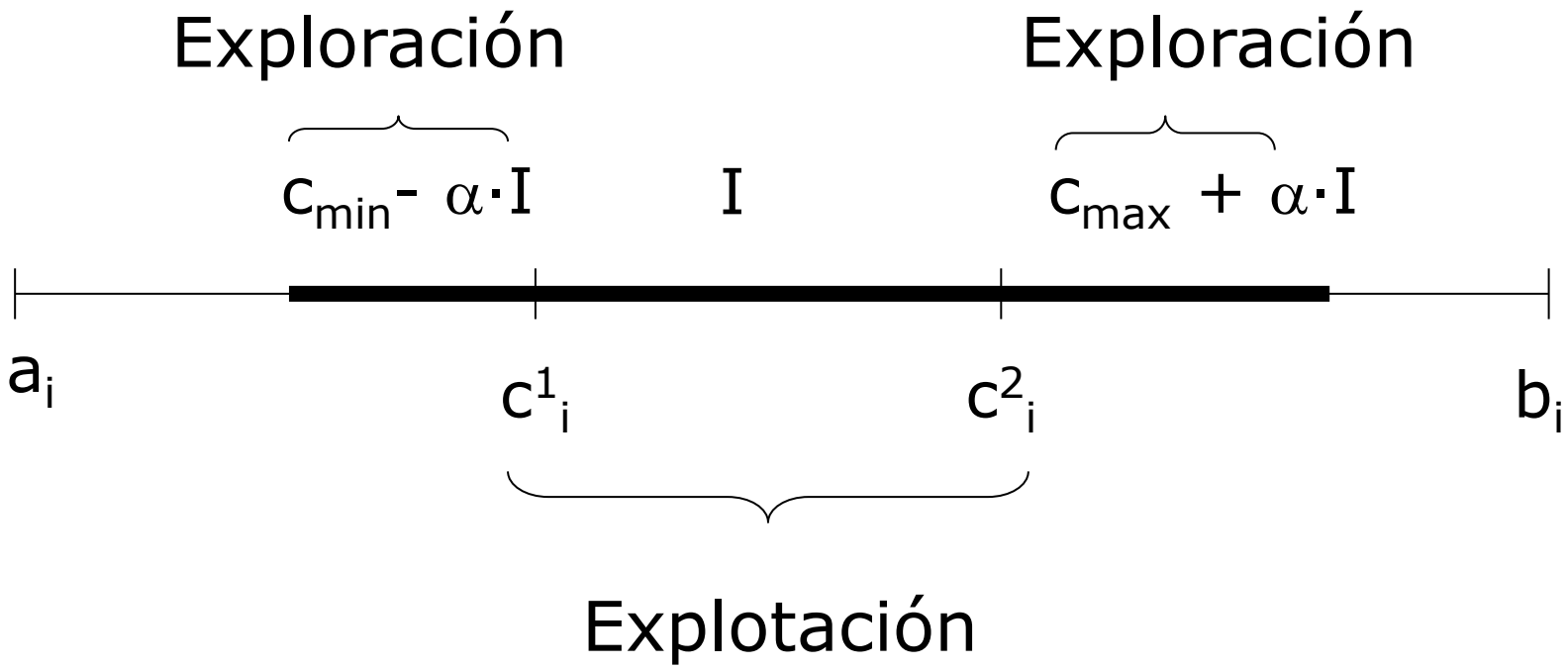
$$H_k = (h_{k1},\ldots, h_{ki},\ldots, h_{kn}) , k = 1,2$$

- donde h_{ki} se genera aleatoriamente en el intervalo:

$$[C_{min} - I\cdot\alpha, C_{max} + I\cdot\alpha]$$

  - $C_{max} = \max \{c_{1i} , c_{2i}\}$
  - $C_{min} = \min \{c_{1i} , c_{2i}\}$
  - $I = C_{max} - C_{min} , \alpha \in [0,1]$

Ejemplo: Operador de cruce para representación real: **BLX-α**



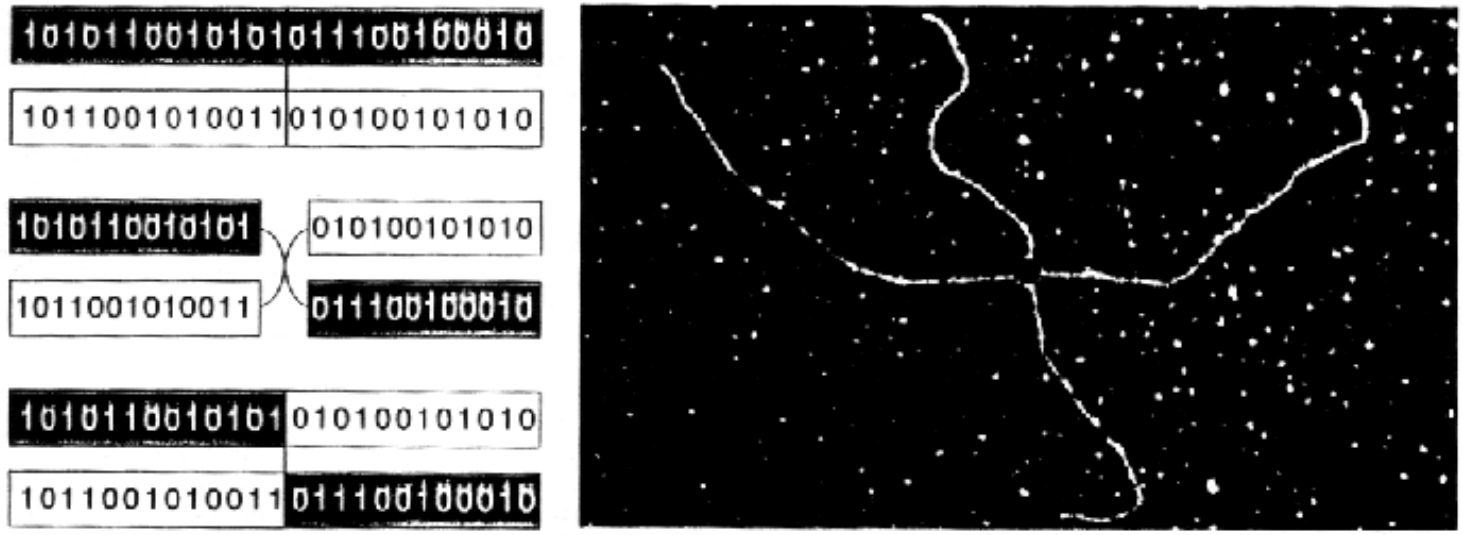
¿Cómo aplicar los algoritmos genéticos a la Representación de Orden?

Padre 1

7	3	1	8	2	4	6	5

Padre 2

4	3	2	8	6	7	1	5

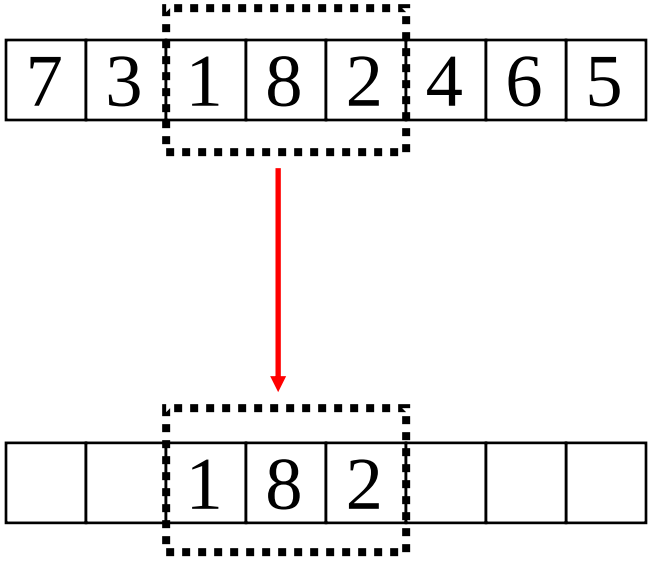


CROSSOVER is the fundamental mechanism of genetic rearrangement for both real organisms and genetic algorithms.

Chromosomes line up and then swap the portions of their genetic code beyond the crossover point.

Representación de Orden y Operador de cruce OX

Padre 1

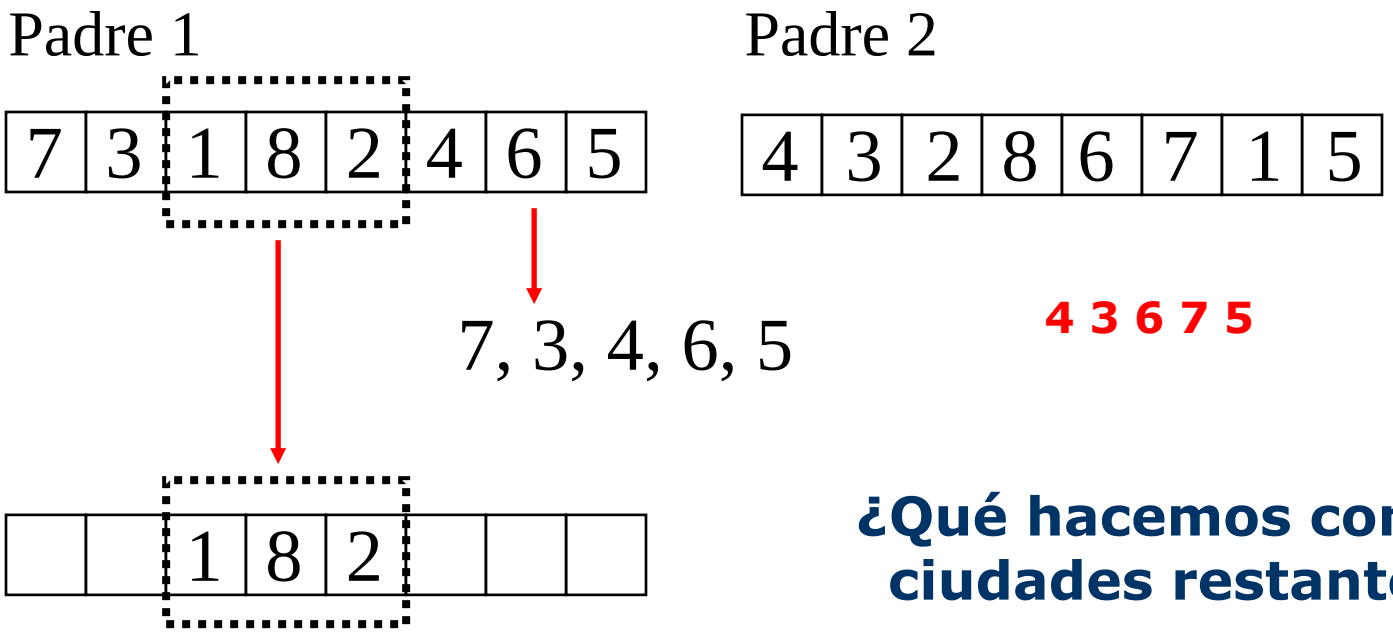


Padre 2

4	3	2	8	6	7	1	5

¿Qué hacemos con las ciudades restantes?

Representación de Orden y Operador de cruce OX

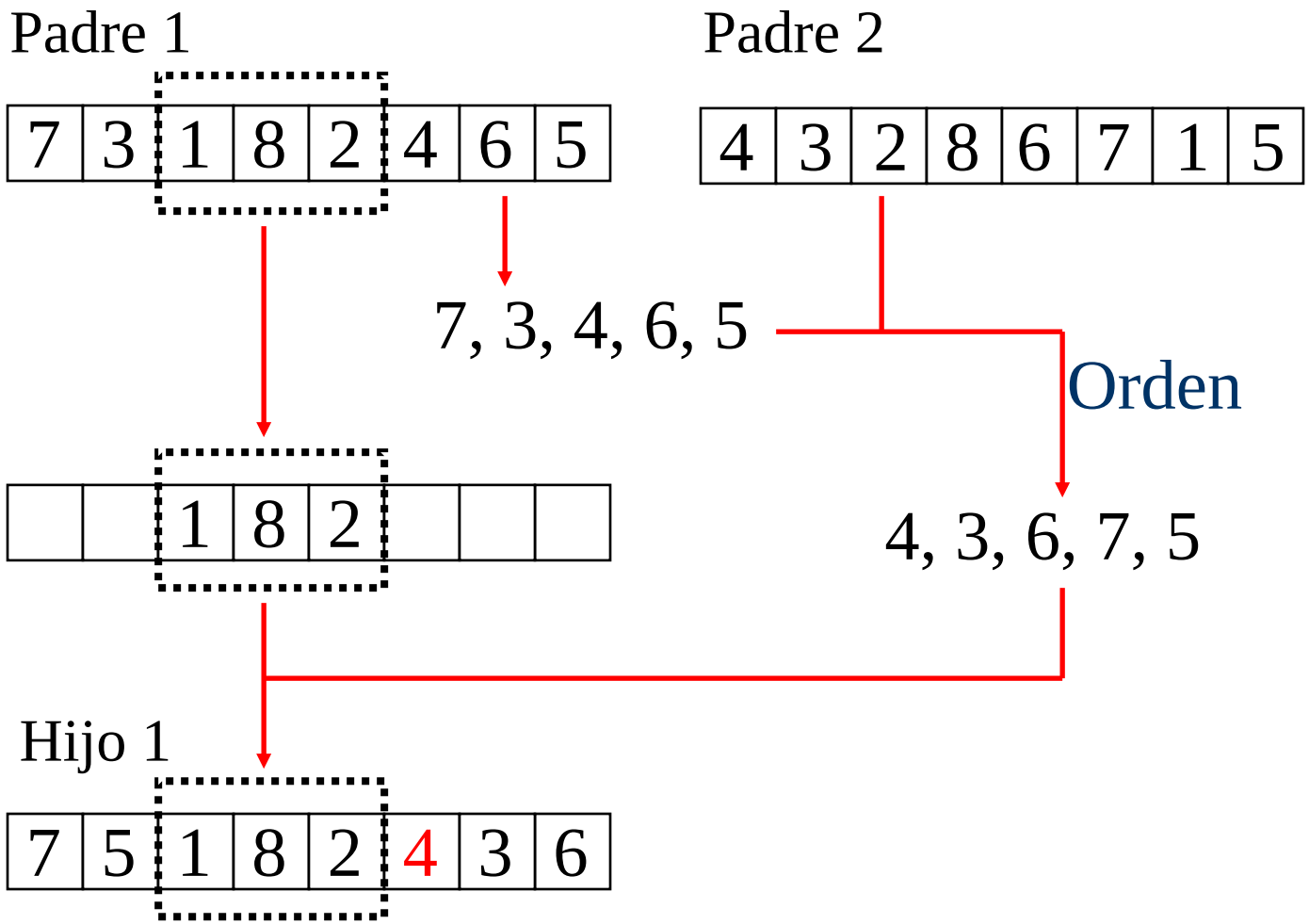


¿Qué hacemos con las ciudades restantes?

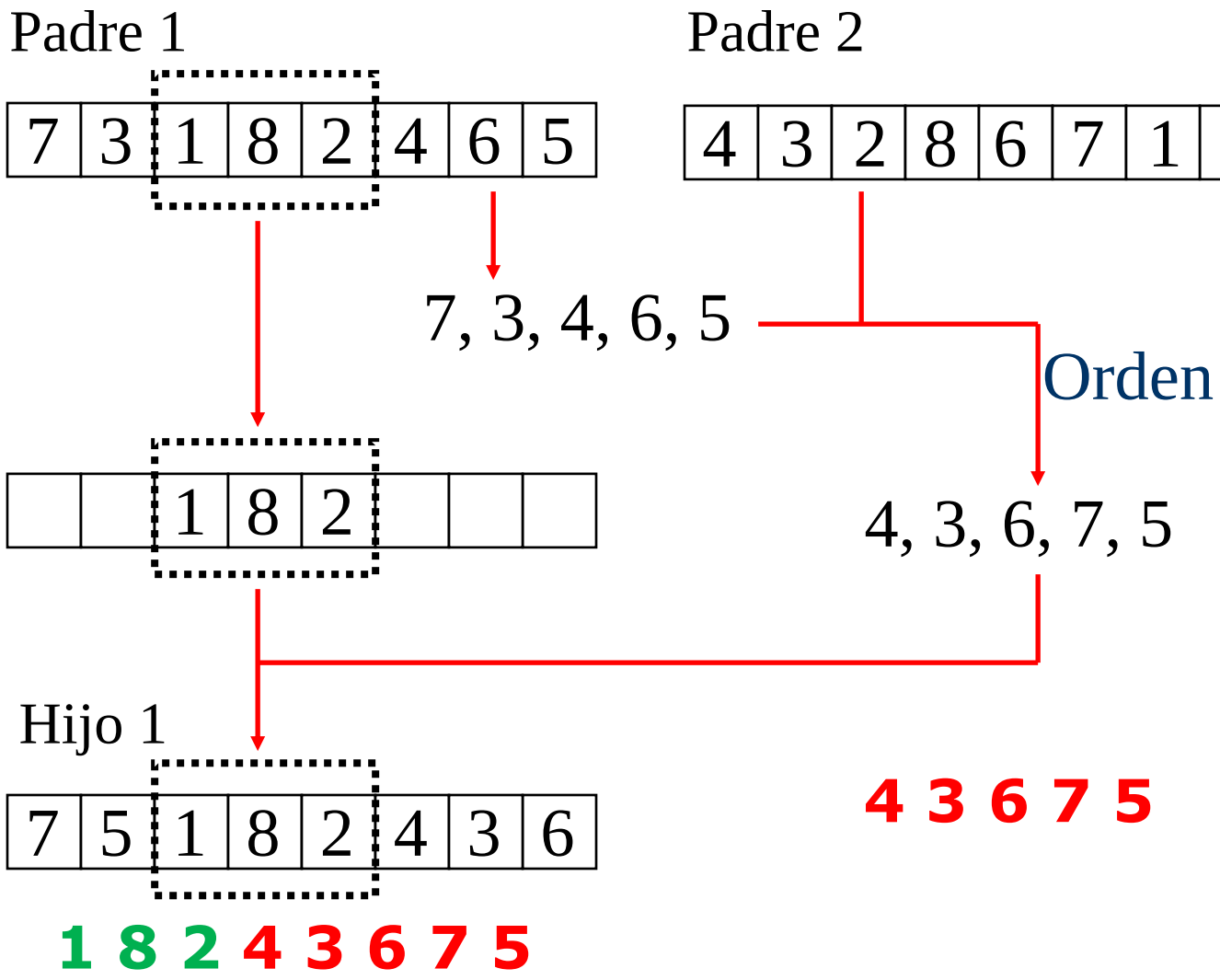
731 86715

Fuera: 4 y 2
Repetidos: 7 y 1

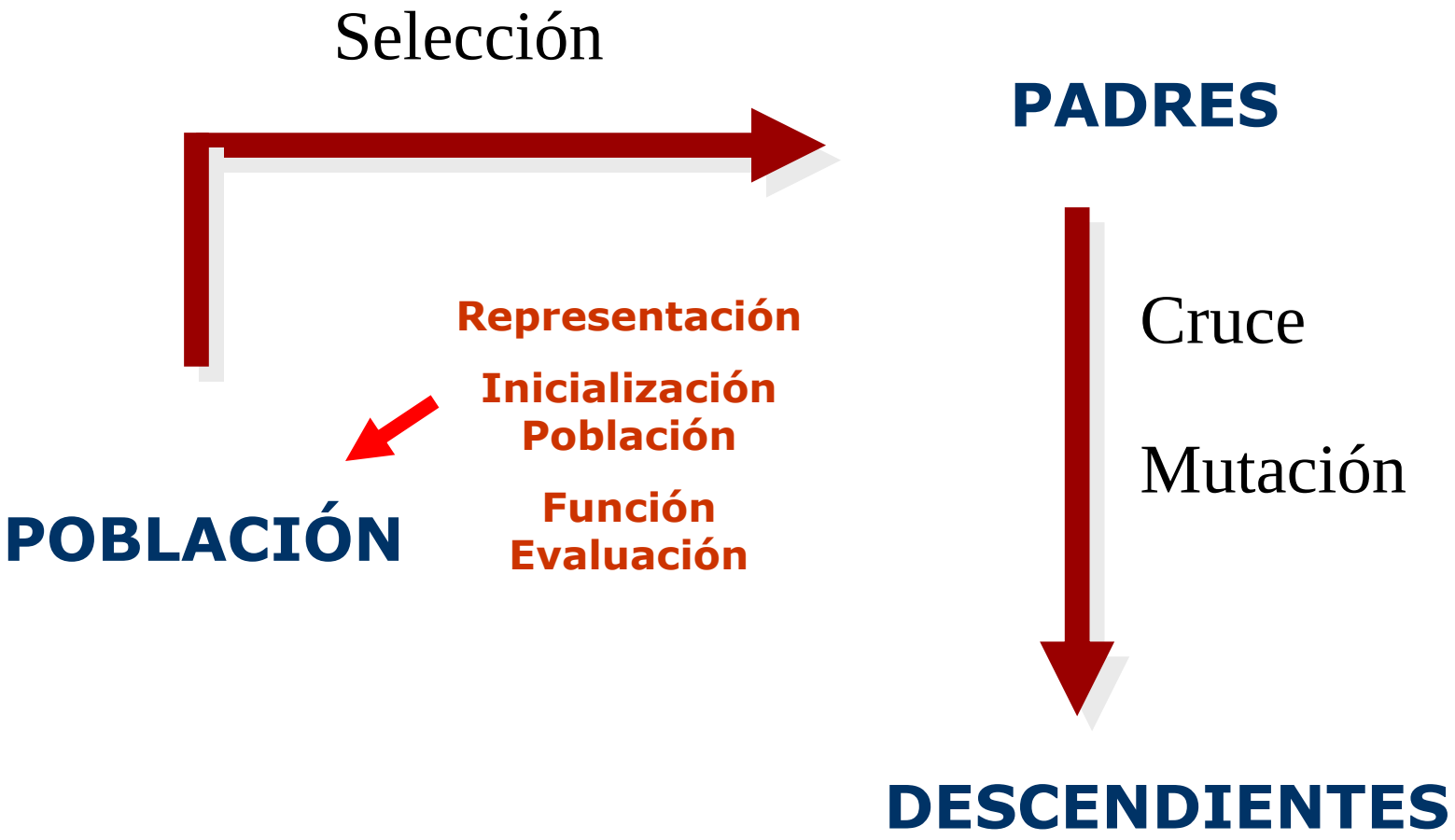
Representación de Orden y Operador de cruce OX



Representación de Orden y Operador de cruce OX



¿CÓMO SE CONSTRUYE UN AG?



Operador de mutación

Podemos tener uno o más operadores de mutación para nuestra representación.

Algunos aspectos importantes a tener en cuenta son:

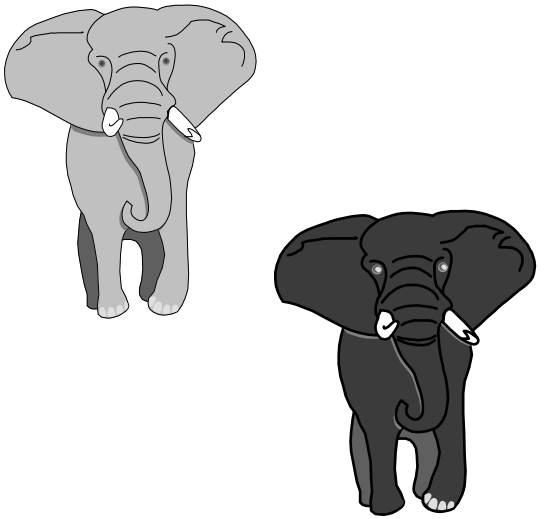
- Debe permitir alcanzar cualquier parte del espacio de búsqueda.
- El tamaño de la mutación debe ser controlado.
- Debe producir cromosomas válidos.
- Se aplica con una probabilidad muy baja de actuación sobre cada descendiente obtenido tras aplicar el operador de cruce (incluidos los descendientes que coinciden con los padres porque el operador de cruce no actúa).

Ejemplo: Mutación para representación discreta binaria

antes 1 1 1 1 1 1 1

después 1 1 1 0 1 1 1

gen mutado



La mutación ocurre con una probabilidad p_m para cada gen

Ejemplo: Mutación para representación real

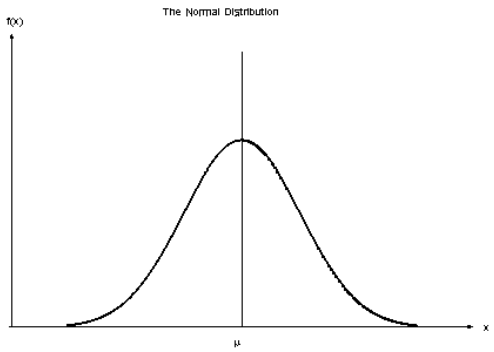
Perturbación de los valores mediante un valor aleatorio.

Frecuentemente, mediante una distribución Gaussiana/normal $N(0,\sigma)$, donde

- 0 es la media
- σ es la desviación típica

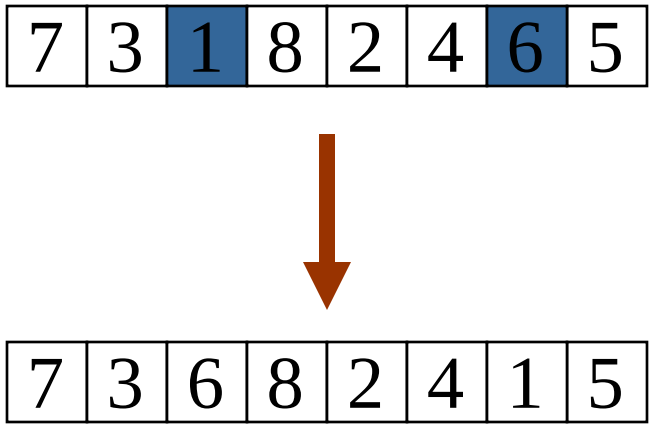
$$x'_i = x_i + N(0,\sigma_i)$$

para cada parámetro.

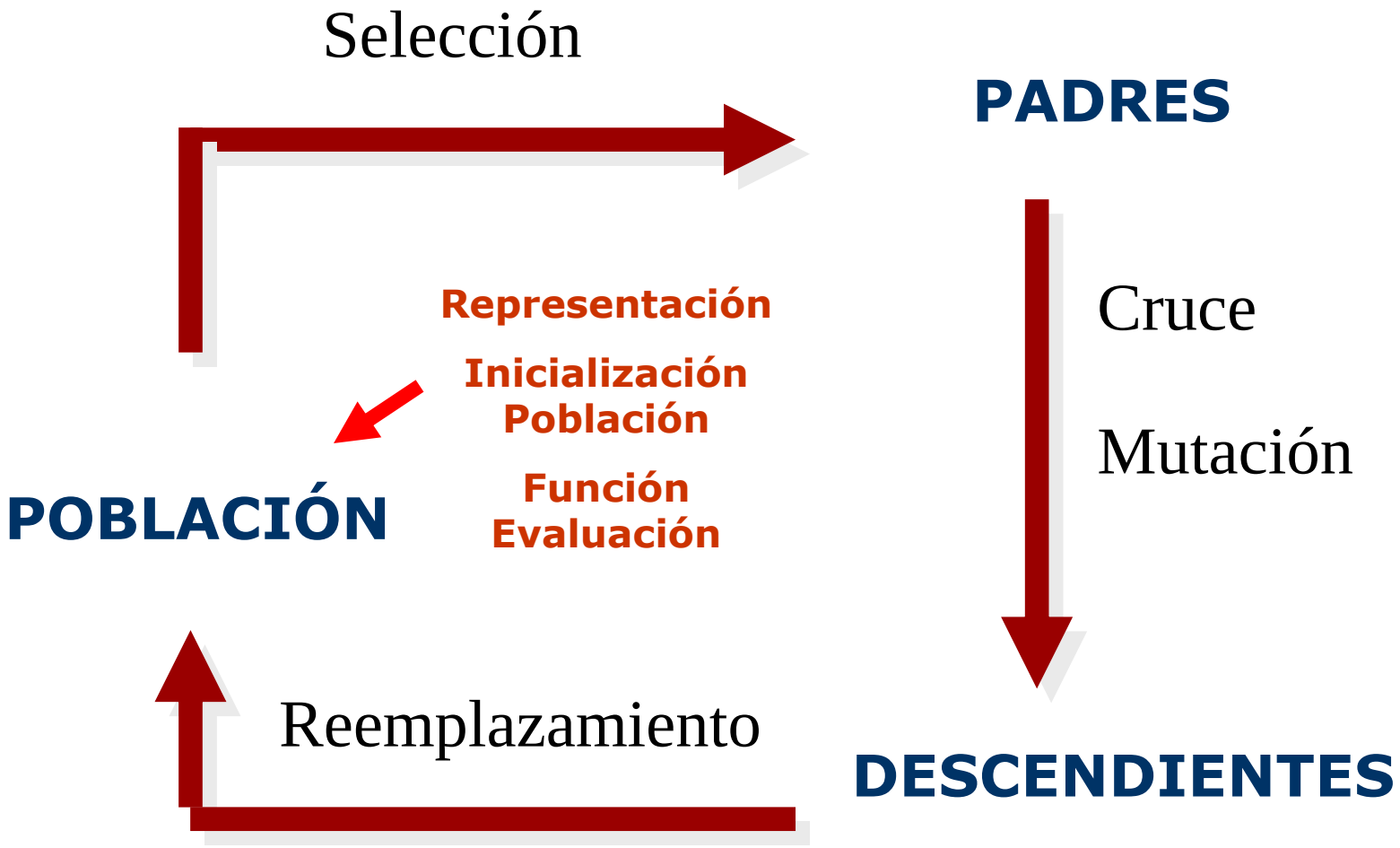


Ejemplo: Mutación para representación de orden

Seleción aleatoria de dos genes e intercambio de ambos.



¿CÓMO SE CONSTRUYE UN AG?



Estrategia de Reemplazamiento

La presión selectiva se ve también afectada por la forma en que los cromosomas de la población son reemplazados por los nuevos descendientes.

Podemos utilizar métodos de reemplazamiento aleatorios, o determinísticos.

Podemos decidir no reemplazar al mejor cromosoma de la población: **Elitismo**

(el uso del Elitismo es aconsejado en los modelos generacionales para no perder la mejor solución encontrada).

Un modelo con alto grado de elitismo consiste en utilizar una población intermedia con todos los padres (N) y todos los descendientes y seleccionar los N mejores. Esto se combina con otras componentes con alto grado de diversidad.

Estrategia de Reemplazamiento

Algunas estrategias de reemplazo para AG estacionarios

Cuando se considera un modelo estacionario (en el que se reemplazan solo uno o dos padres, frente al modelo generacional en el que se reemplaza la población completa), nos encontramos con diferentes propuestas.

A continuación presentamos algunas posibilidades:

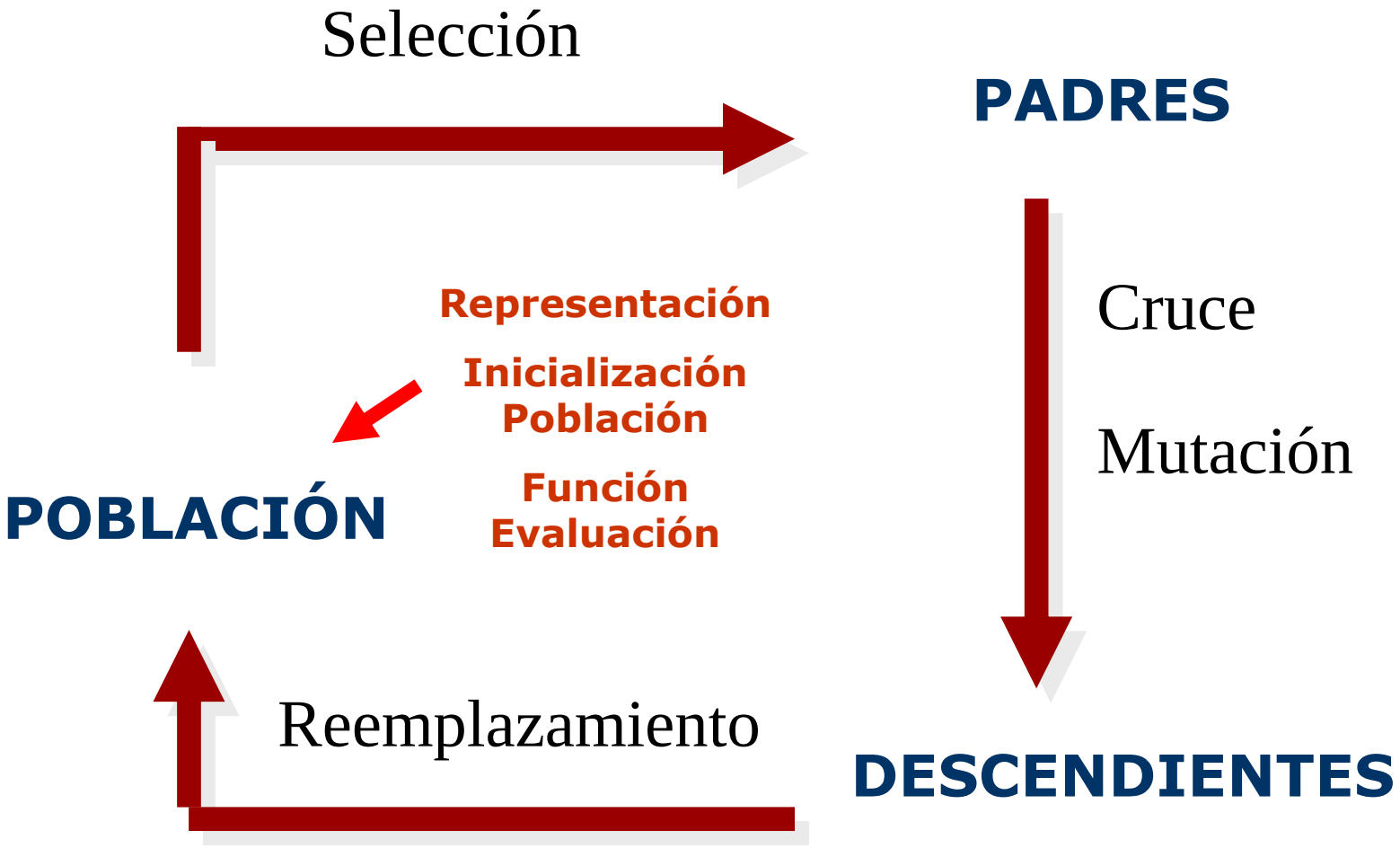
- Reemplazar al peor de la población (RW). Genera alta presión selectiva.
- Torneo Restringido (RTS): Se reemplaza al mas parecido de entre *w (w=3, …)*. Mantiene una cierta diversidad.
- Peor entre semejantes (WAMS): Se reemplaza el peor cromosoma del conjunto de los *w (w=3, …)* padres más parecidos al descendiente generado (seleccionados de toda la población). Busca equilibrio entre diversidad y presión selectiva.
- Algoritmo de Crowding Determinístico (DC): El hijo reemplaza a su padre más parecido. Mantiene diversidad.

Criterio de Parada

- Cuando se alcanza el óptimo!
- Recursos limitados de CPU:

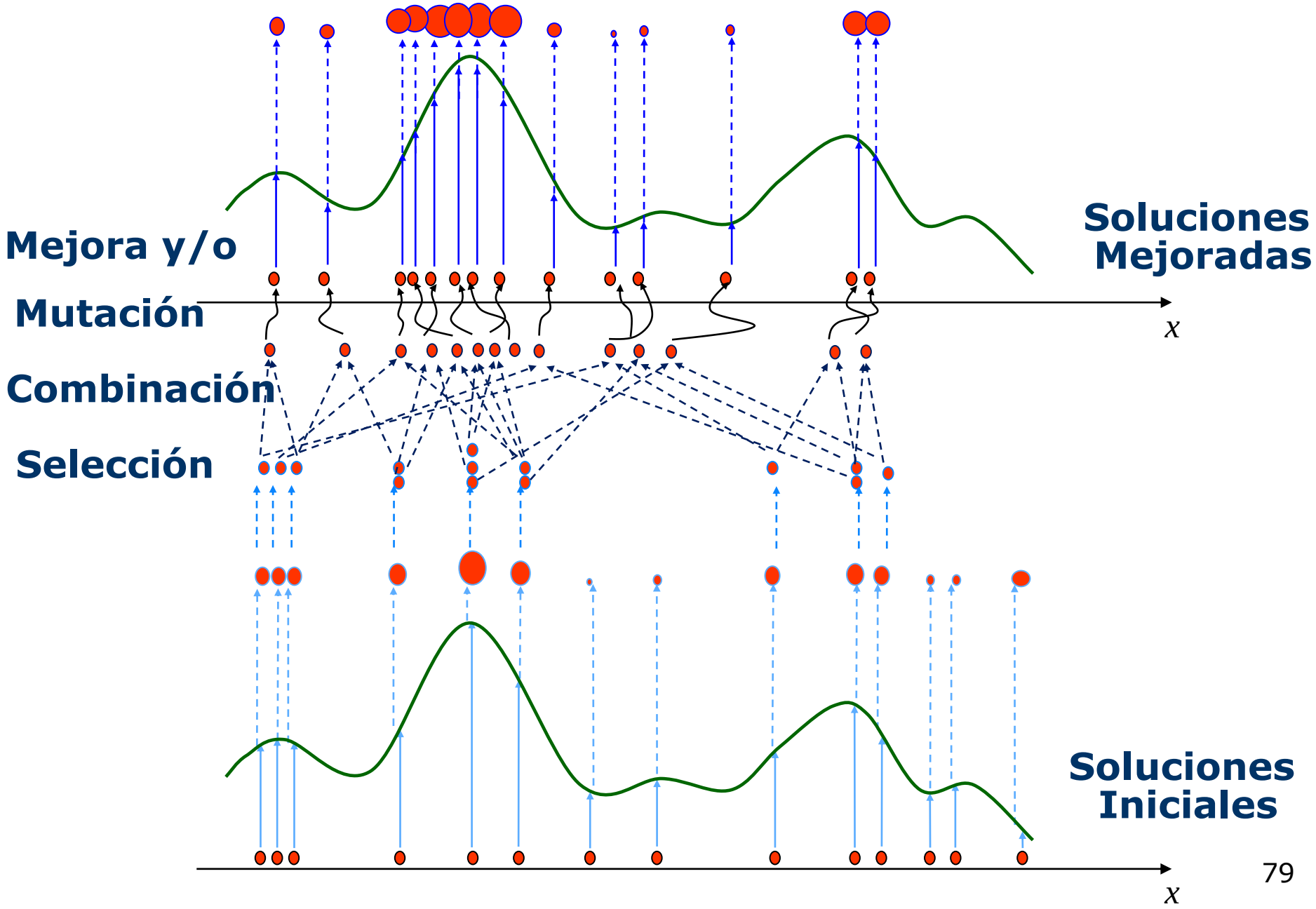
  Fijar el máximo número de evaluaciones
- Límite sobre la paciencia del usuario: Después de algunas iteraciones sin mejora.

¿CÓMO SE CONSTRUYE UN AG? RESUMEN



PROCESO ITERATIVO + CRITERIO DE PARADA

Efecto de la Evolución



4. SOBRE SU UTILIZACIÓN

- Nunca se deben sacar conclusiones de una única ejecución
  - **utilizar medidas estadísticas (medias, medianas, ...)**
  - **con un número suficiente de ejecuciones independientes**
- No se debe ajustar/chequear la actuación de un algoritmo sobre ejemplos simples si se desea trabajar con casos reales.
- Existe una comentario genérico en el uso de los Algoritmos no determinísticos:

  "Se puede obtener lo que se desea en una experimentación de acuerdo a la dificultad de los casos utilizados"

  (se encuentran propuestas en las que basta encontrar un caso adecuado para un algoritmo para afirmar que es muy bueno, pero esta afirmación no puede ser extensible a otros casos, es el error en el que incurren algunos autores)

5. EJEMPLO: VIAJANTE DE COMERCIO

Representación de orden

(3 5 1 13 6 15 8 2 17 11 14 4 7 9 10 12 16)

17 ciudades

Objetivo: Suma de la distancia entre las ciudades.

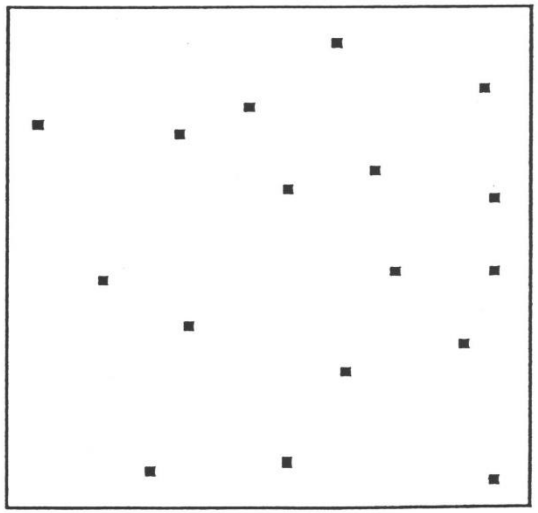
Población: 61 cromosomas - Elitismo

Cruce: OX (P_c = 0.6) **(30 parejas, 18 cruzan, 36 descendientes nuevos por iteración)**

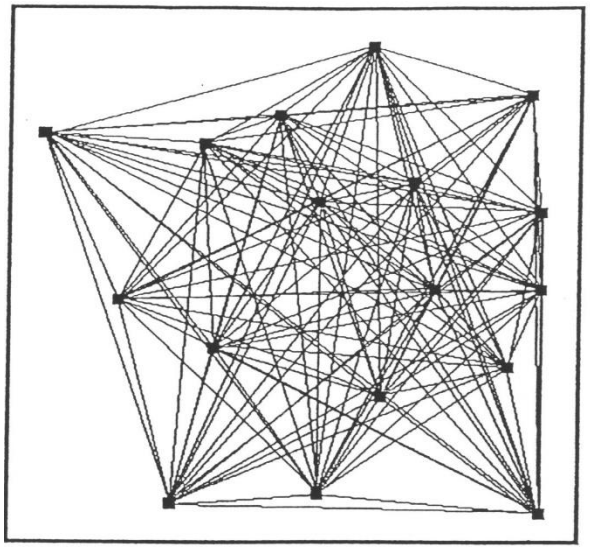
Mutación: Inversión de una lista (P_m = 0.01 – cromosoma)

Viajante de Comercio

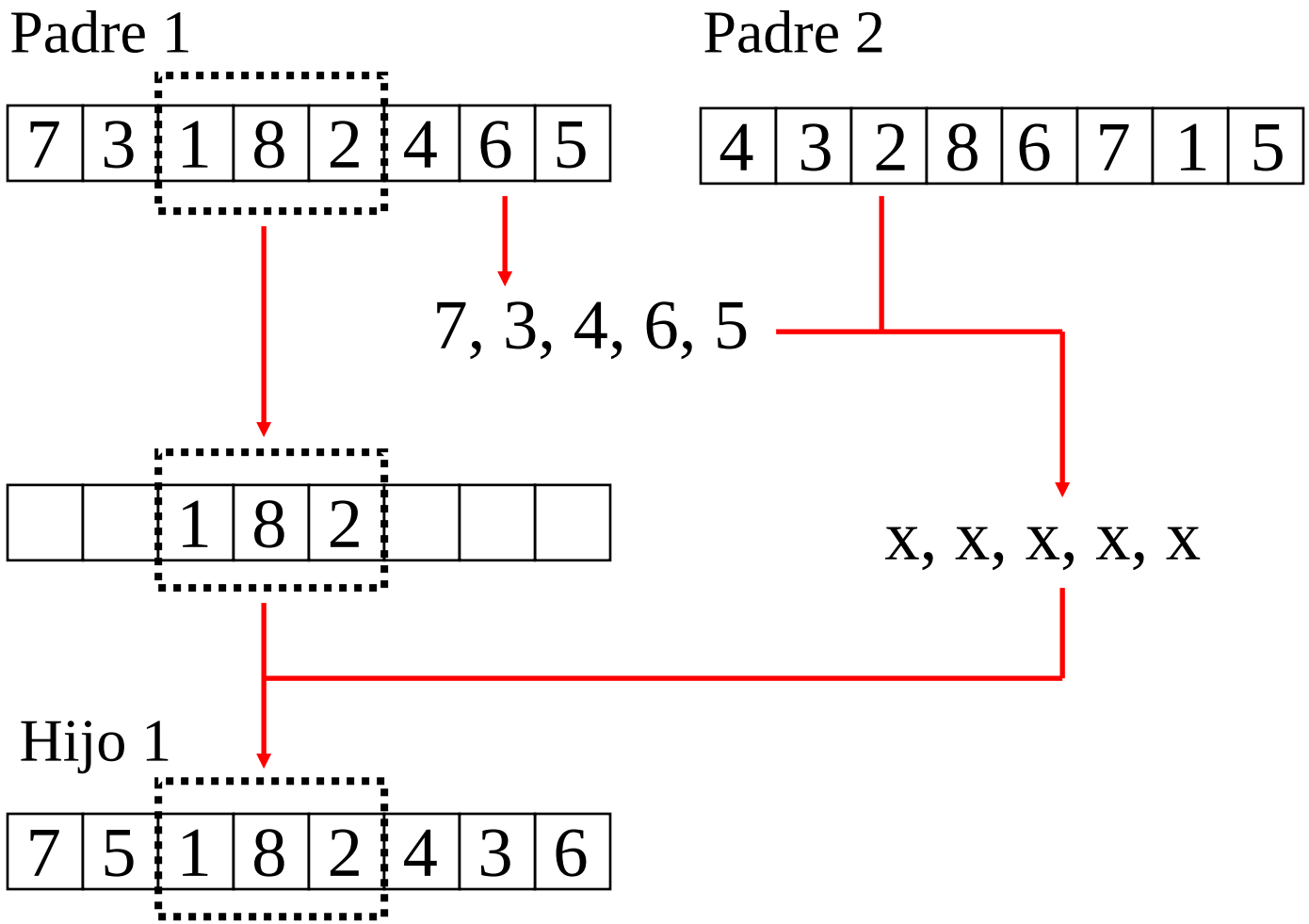
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	12.87	19.71	31.56	22.70	17.26	23.33	12.16	24.71	34.51	12.58	21.38	42.37	27.43	36.51	19.10	1.18
2	0	15.80	37.51	21.52	28.57	35.43	22.70	16.78	28.57	11.13	25.26	50.62	38.16	35.97	9.04	34.56	
3	0	50.18	36.56	35.86	35.51	21.60	31.50	43.51	25.58	38.78	61.57	46.15	51.10	23.50	48.52		
4	0	20.90	21.52	37.62	38.14	33.26	31.90	27.13	13.03	15.53	18.39	19.37	35.84	8.12			
5	0	26.00	40.72	33.74	12.87	14.71	11.68	9.72	35.86	30.96	15.06	16.78	15.27				
6	0	16.99	18.53	34.51	40.20	22.34	18.53	27.70	10.80	34.94	32.08	25.24					
7	0	14.54	46.60	54.54	33.80	34.52	40.35	22.09	51.20	41.84	41.73						
8	0	36.31	46.12	24.21	30.50	45.72	28.09	46.77	30.20	39.71							
9	0	12.54	13.31	21.52	48.18	41.50	23.85	8.50	27.43								
10	0	22.43	23.33	46.67	44.80	16.31	20.53	24.58									
11	0	14.71	40.81	30.52	26.21	10.50	23.93										
12	0	27.43	21.97	17.20	23.35	10.35											
13	0	18.89	32.78	50.15	22.59												
14	0	35.88	40.51	24.71													
15	0	30.18	11.90														
16	0	31.31															
17	0																



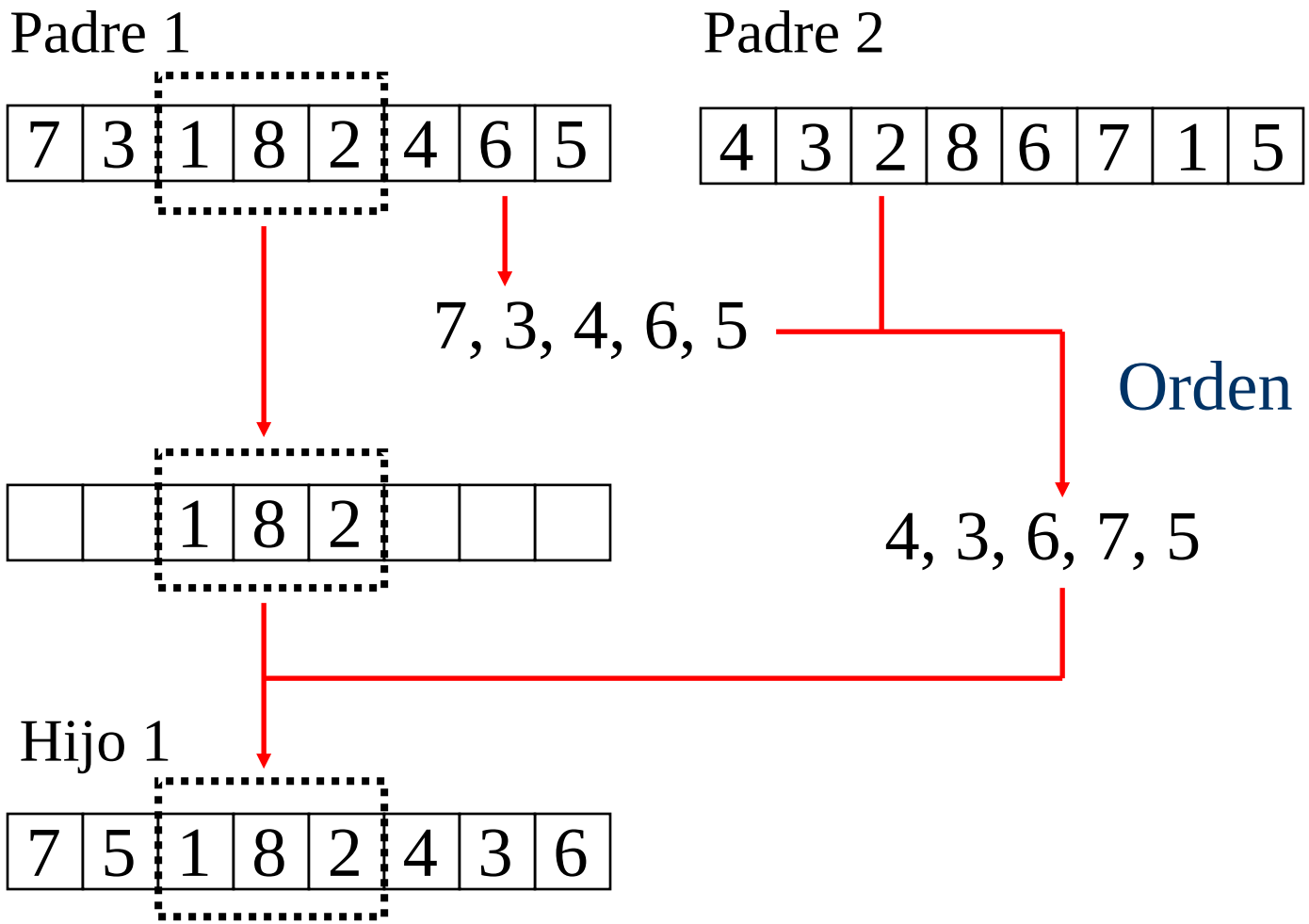
17! (3.5568734e14) soluciones posibles



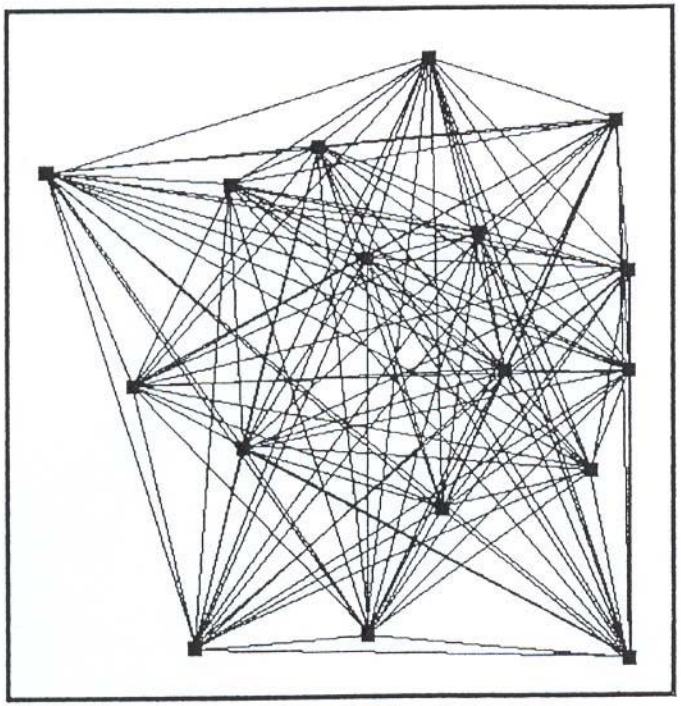
Representación de Orden y Operador de cruce OX

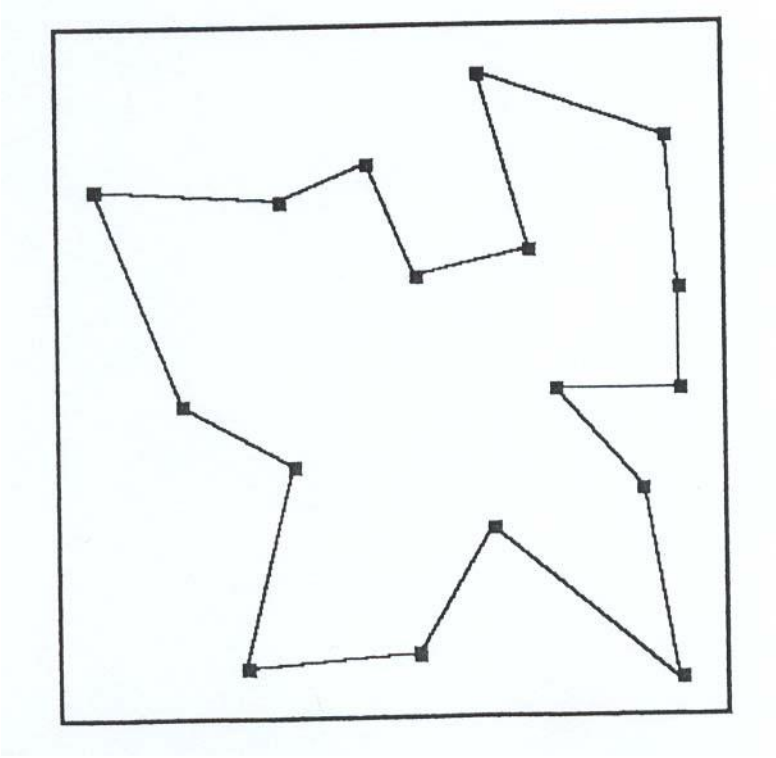


Representación de Orden y Operador de cruce OX



Viajante de Comercio



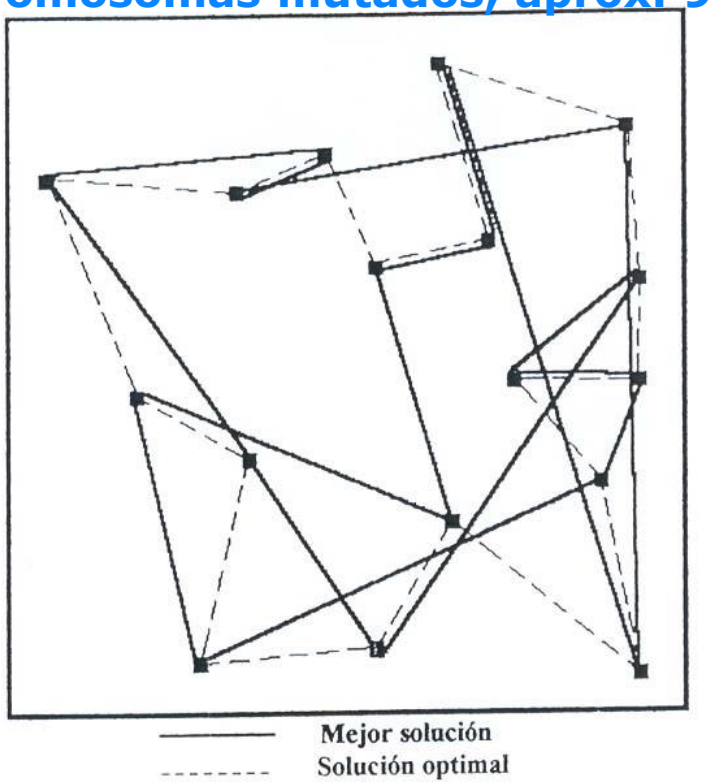


17! = 3.5568743 e14 recorridos posibles

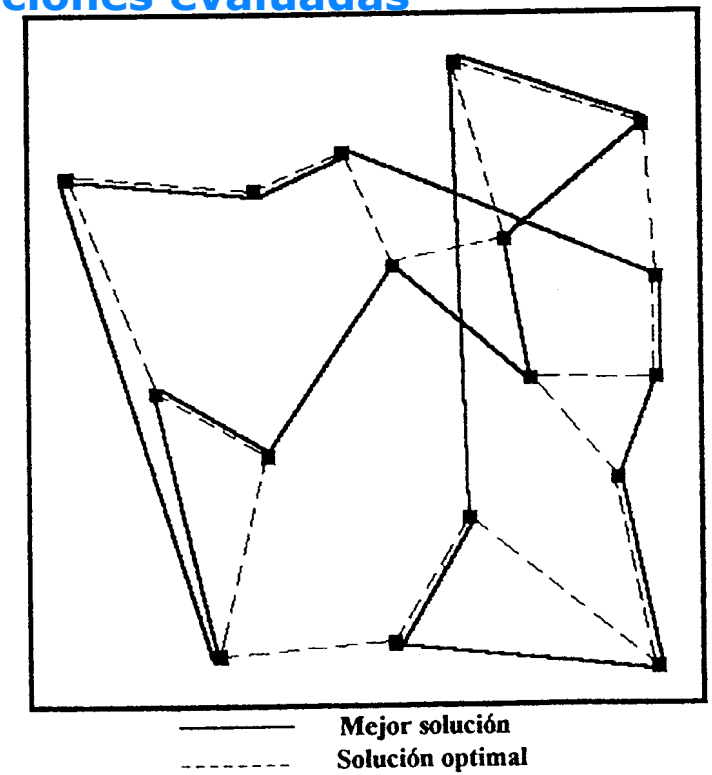
Solución óptima: 226.64

Viajante de Comercio

36 descendientes nuevos por iteración x 25 = 900 descendientes, más 15 cromosomas mutados, aprox. 906 soluciones evaluadas



Iteración: 0 Costo: 403.7

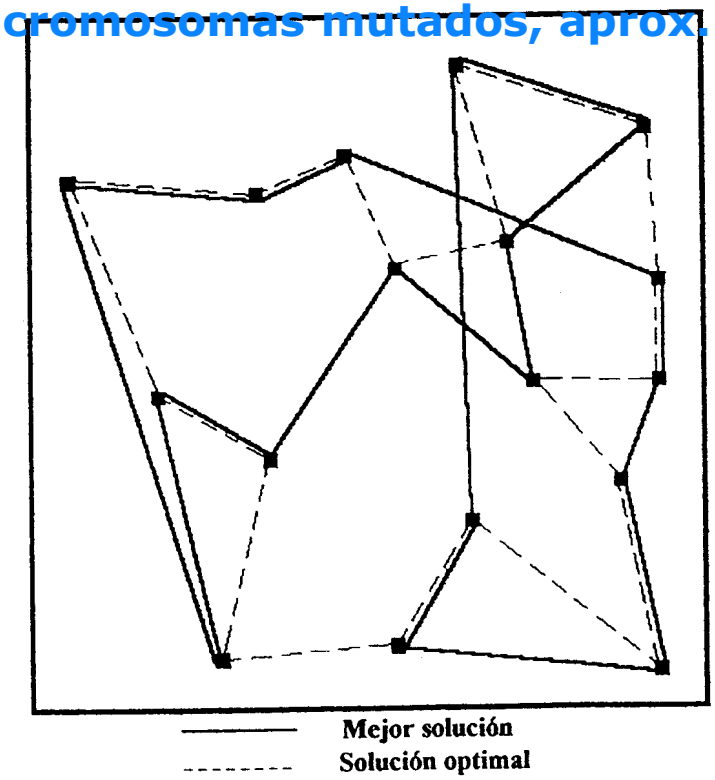


Iteración: 25 Costo: 303.86

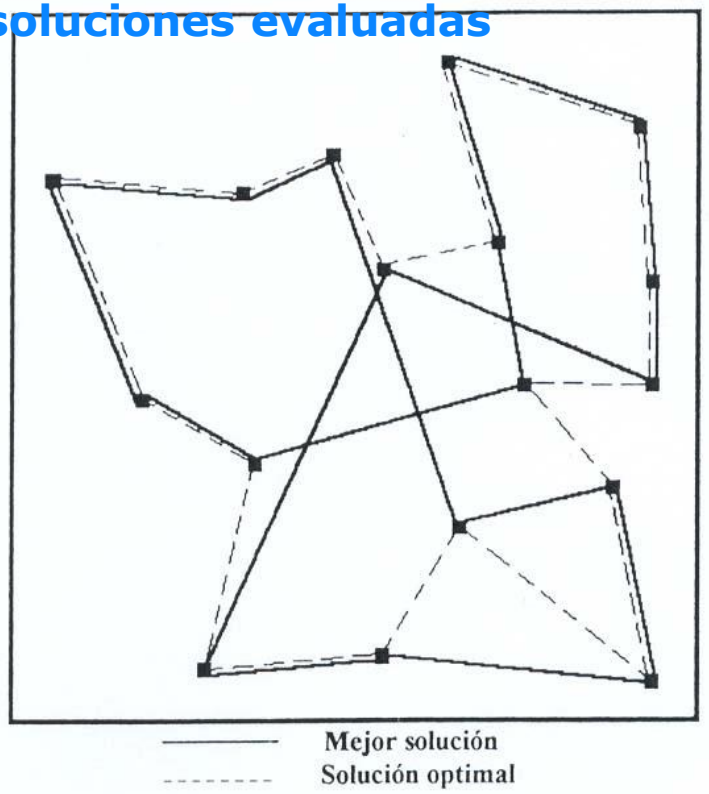
Solución óptima: 226.64

Viajante de Comercio

36 descendientes nuevos por iteración x 50 = 1800 descendientes, más 30 cromosomas mutados, aprox. 1812 soluciones evaluadas



Iteración: 25 Costo: 303.86

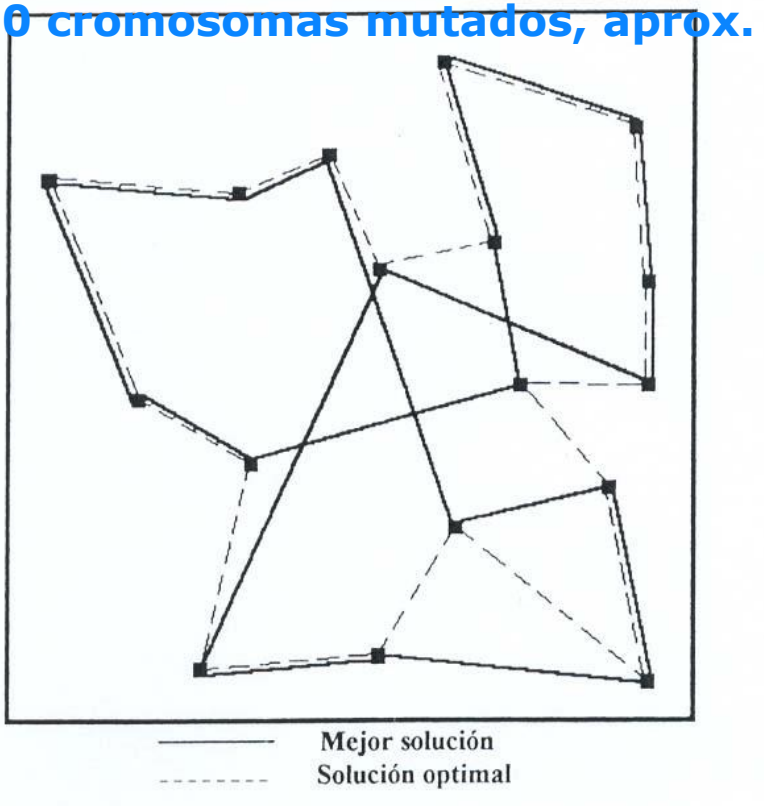


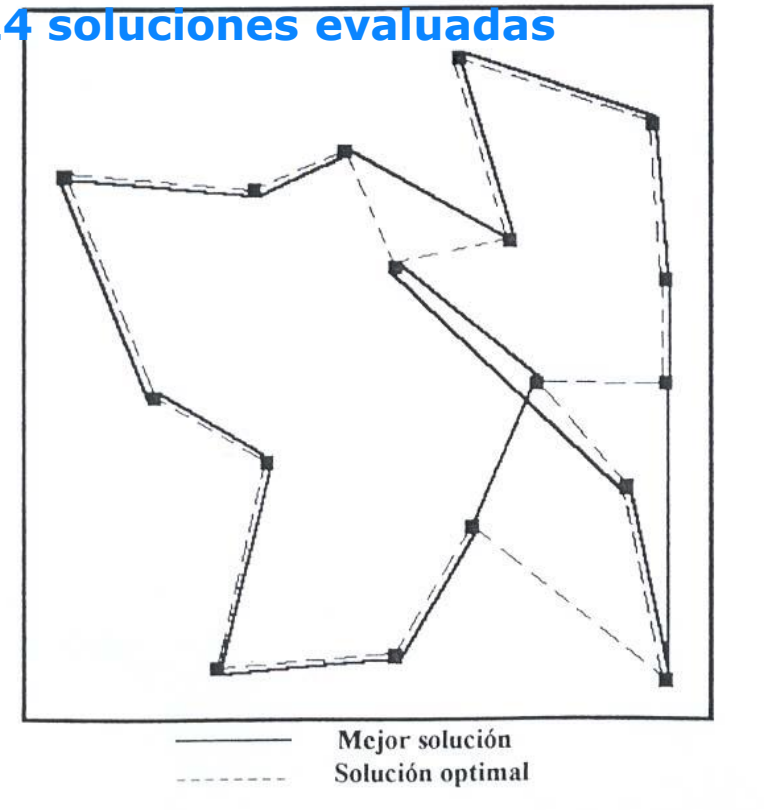
Iteración: 50 Costo: 293.6

Solución óptima: 226.64

Viajante de Comercio

36 descendientes nuevos por iteración x 100 = 3600 descendientes, más 60 cromosomas mutados, aprox. 3624 soluciones evaluadas





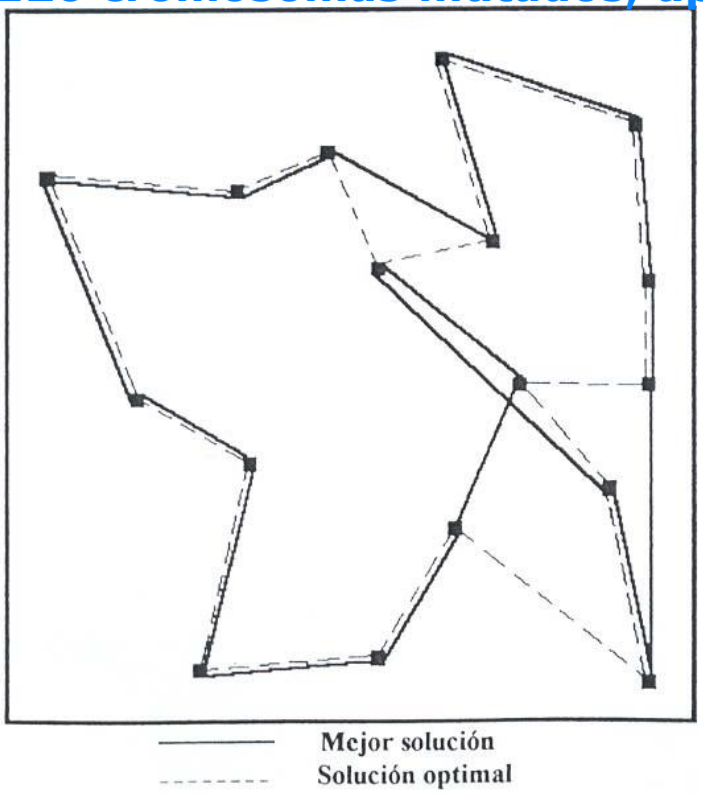
Iteración: 50 Costo: 293.6

Iteración: 100 Costo: 256.55

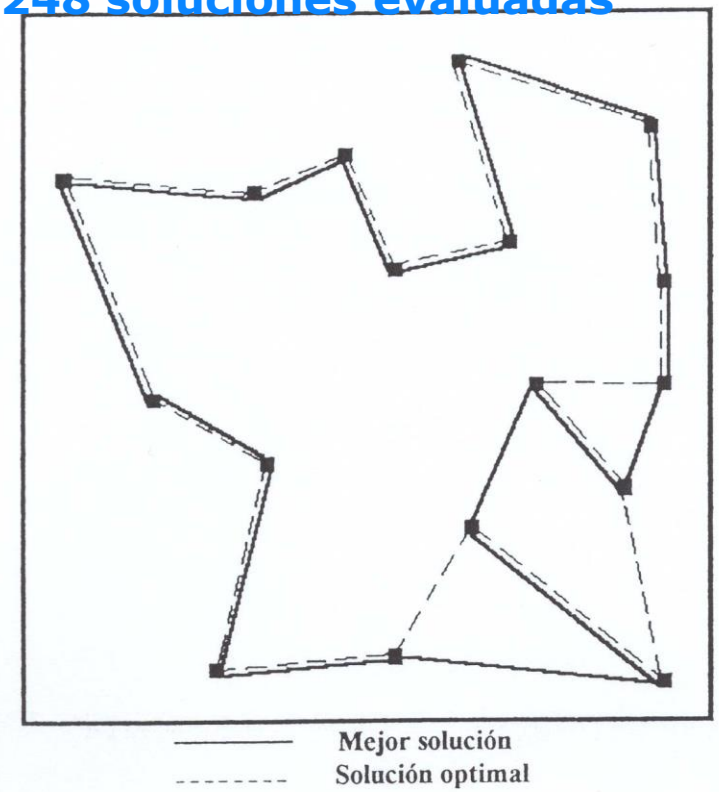
Solución óptima: 226.64

Viajante de Comercio

36 descendientes nuevos por iteración x 200 = 7200 descendientes, más 120 cromosomas mutados, aprox. 7248 soluciones evaluadas



Iteración: 100 Costo: 256.55

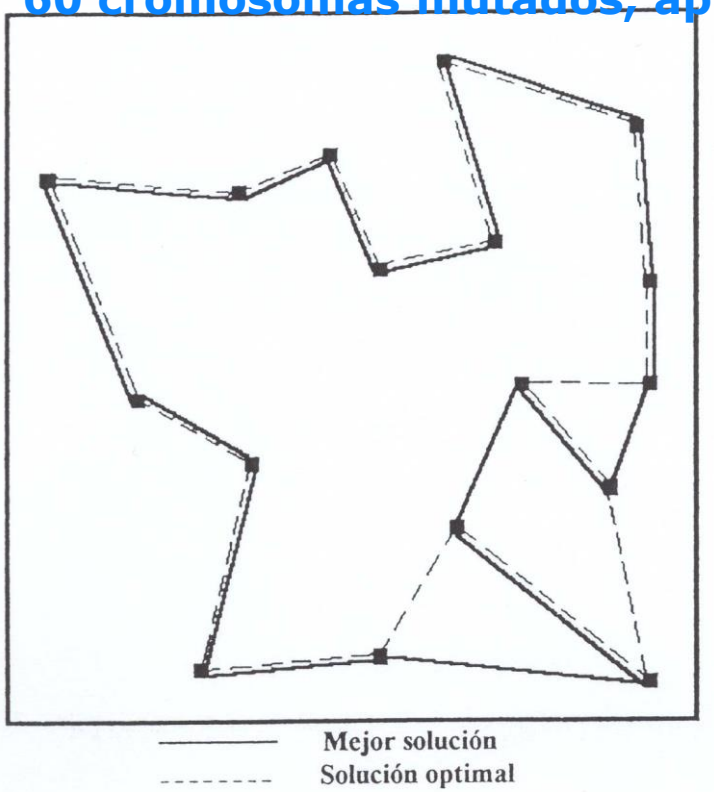


Iteración: 200 Costo: 231.4

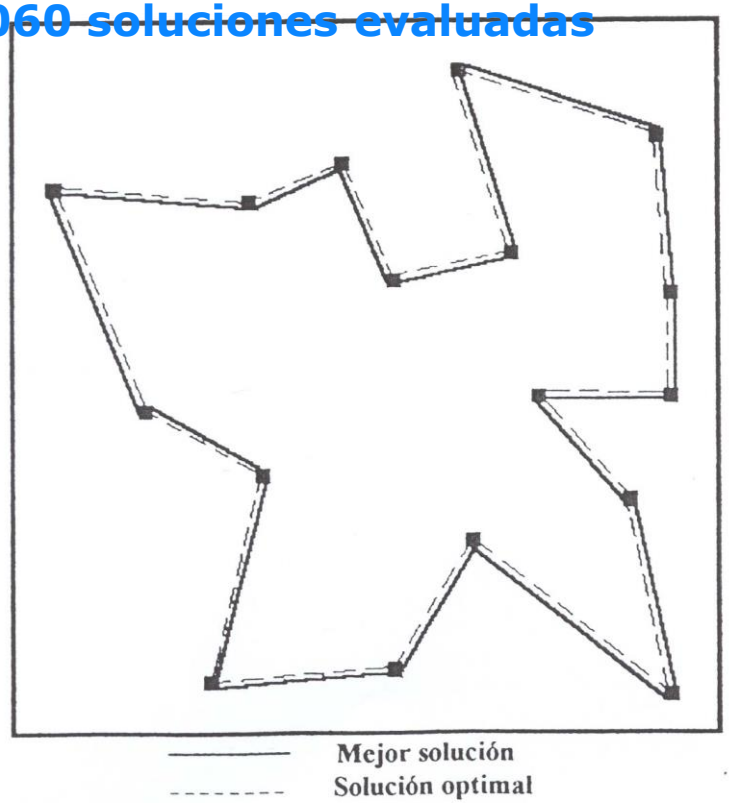
Solución óptima: 226.64

Viajante de Comercio

36 descendientes nuevos por iteración x 250 = 9000 descendientes, más 60 cromosomas mutados, aprox. 9.060 soluciones evaluadas



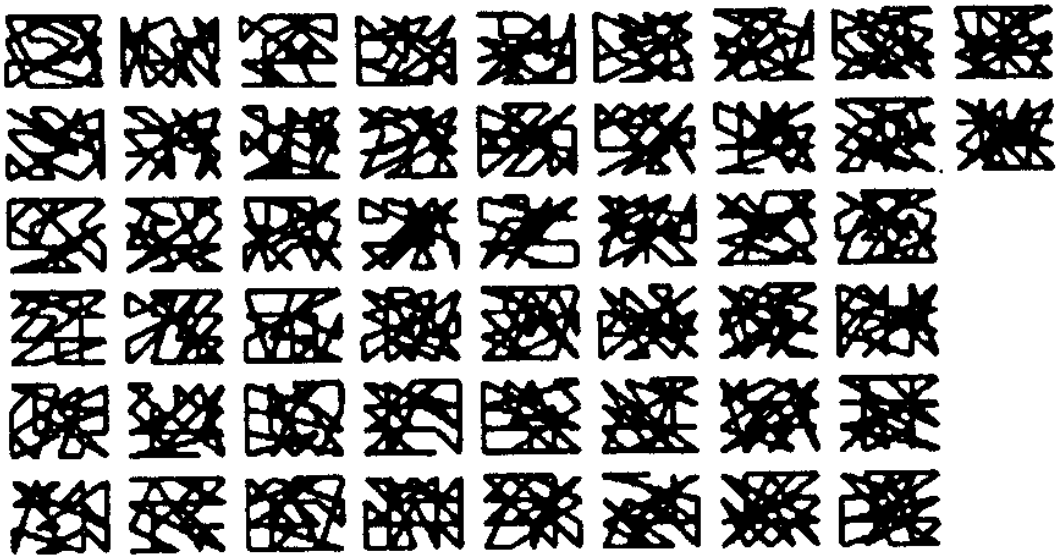
Iteración: 200 Costo: 231.4



Iteración: 250 Solución óptima: 226.64

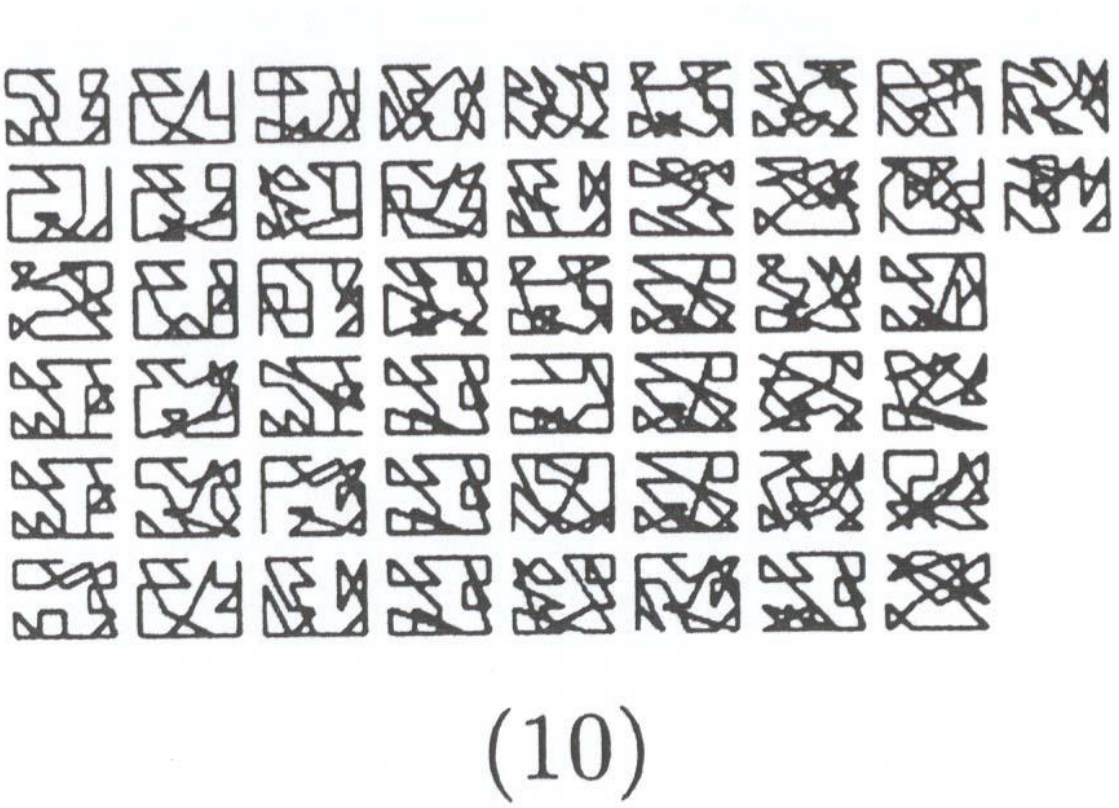
17! = 3.5568743 e14 recorridos posibles

Viajante de Comercio



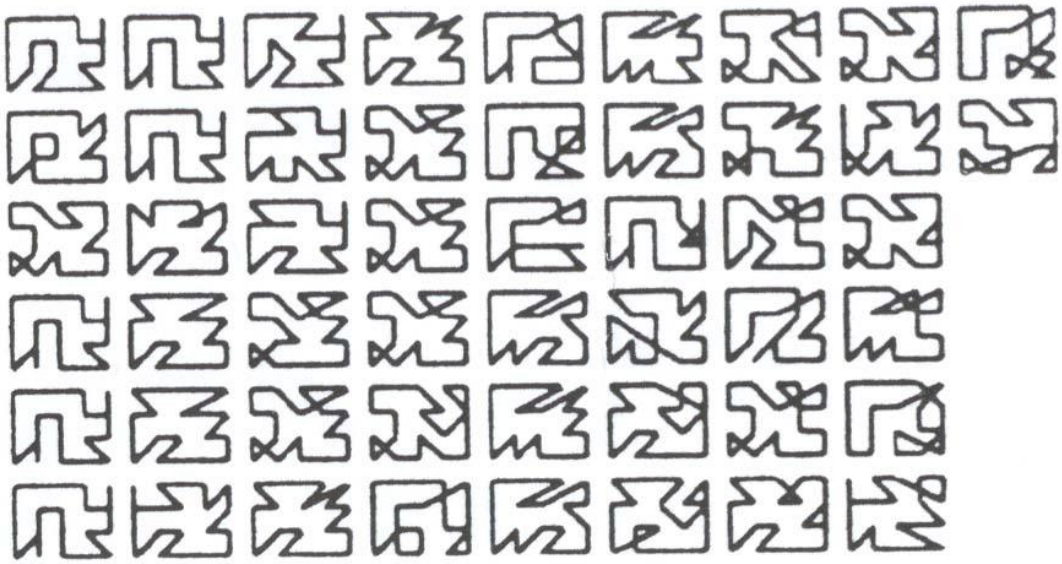
Visualización de la evolución de una población de 50 cromosomas y 70 iteraciones

Viajante de Comercio



Visualización de la evolución de una población de 50 cromosomas y 70 iteraciones

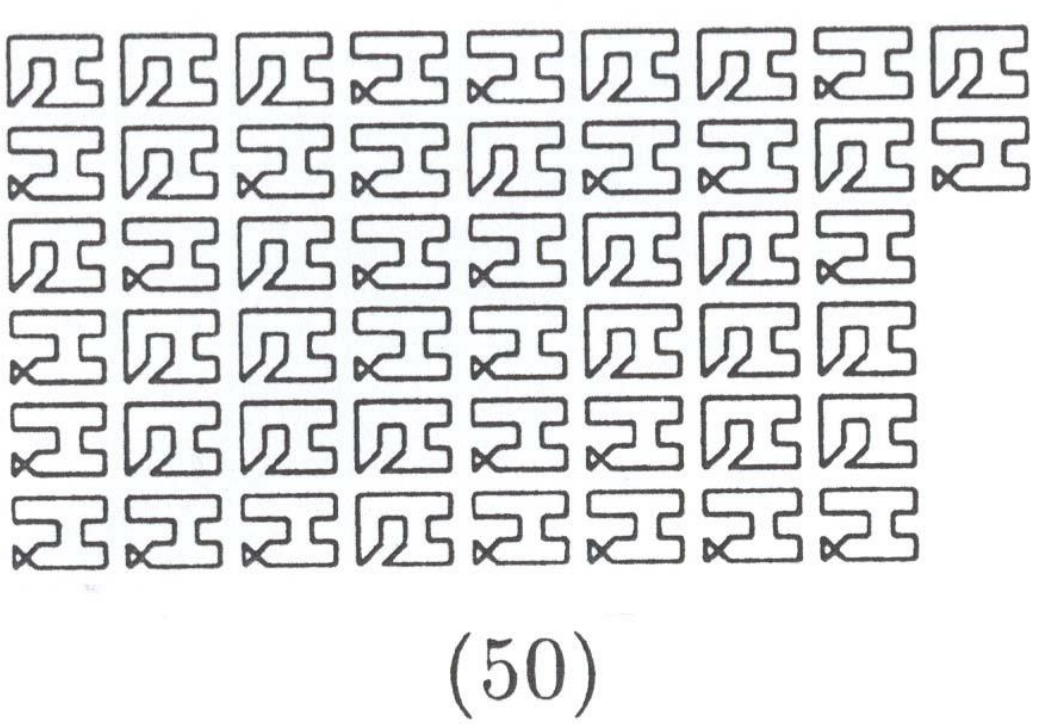
Viajante de Comercio



(30)

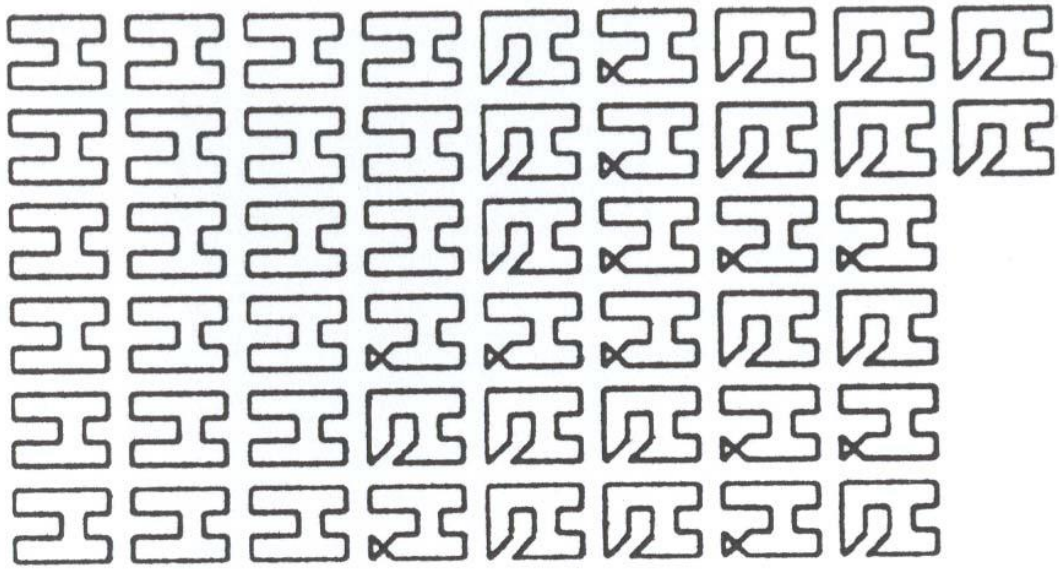
Visualización de la evolución de una población de 50 cromosomas y 70 iteraciones

Viajante de Comercio



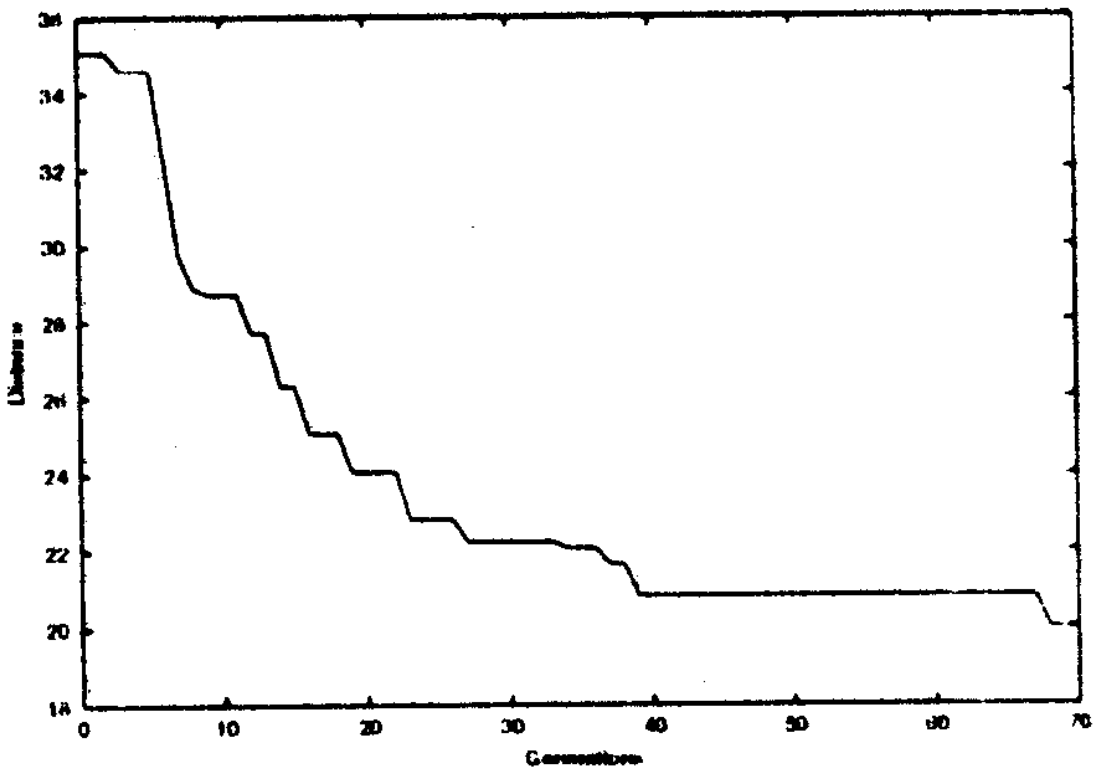
Visualización de la evolución de una población de 50 cromosomas y 70 iteraciones

Viajante de Comercio



Visualización de la evolución de una población de 50 cromosomas y 70 iteraciones

Viajante de Comercio



Visualización de la evolución de una población de 50 cromosomas y 70 iteraciones

COMENTARIOS FINALES

Algoritmos Genéticos

- basados en una metáfora biológica: evolución
- gran potencialidad de aplicación
- muy populares en muchos campos
- muy potentes en diversas aplicaciones
- altas prestaciones a bajo costo

➢ **SON ATRACTIVOS DESDE UN PUNTO DE VISTA COMPUTACIONAL**

Algoritmos Genéticos: Extensiones, estudios, modelos, ...

Algoritmos Genéticos I. Diversidad y Convergencia (Tema 7)

Algoritmos Genéticos II. Múltiples Soluciones en Problemas Multimodales (Tema 7)

Algoritmos Genéticos III. Manejo de restricciones (Seminario)

Algoritmos Evolutivos para Problemas Multiobjetivo (Seminario)

Hibridación de los Algoritmos Evolutivos con Técnicas de Búsqueda Local: Algoritmos Meméticos (Tema 4)

CONCLUSIONES

Los algoritmos evolutivos son la metaheurística que más se ha estudiado y con mayor número de propuestas.

Presenta un mayor número de variantes y posibilidades de aplicación (multimodalidad, multi-objetivos, soluciones de longitud variable …).

En la actualidad existen diferentes áreas de desarrollo muy activas y con propuestas que continuamente aportan nuevas soluciones a los problemas planteados.

La hibridación con algoritmos basados en trayectorias ha dado lugar a interesantes propuestas (Tema 4: Metaheurísticas Híbridas).

METAHEURÍSTICAS

TEMA 3. METAHEURÍSTICAS BASADAS EN POBLACIONES

Parte II:

1. ALGORITMOS EVOLUTIVOS PARA OPTIMIZACIÓN CONTINUA
2. ALGORITMOS GENÉTICOS PARA OPT. CONTINUA
3. EVOLUCIÓN DIFERENCIAL
4. ESTRATEGIAS DE EVOLUCIÓN
5. NUEVOS MODELOS BIOINSPIRADOS PARA OPTIMIZACIÓN DE PARÁMETROS

BIBLIOGRAFÍA
D.B. Fogel (Ed.). Evolutionary Computation. The Fossil Record. (Selected Readings on the History of Evolutionary Computation). IEEE Press, 1998.
A.E. Eiben, J.E. Smith. Introduction to Evolutionary Computation. Springer Verlag 2003.

METAHEURÍSTICAS

2021 - 2022

- Tema 1. Introducción a las Metaheurísticas
- Tema 2. Modelos de Búsqueda: Entornos y Trayectorias vs Poblaciones
- Tema 3. Metaheurísticas Basadas en Poblaciones
- Tema 4: Algoritmos Meméticos
- Tema 5. Metaheurísticas Basadas en Trayectorias
- Tema 6. Metaheurísticas Basadas en Adaptación Social
- Tema 7. Aspectos Avanzados en Metaheurísticas
- Tema 8. Metaheurísticas Paralelas