

### ■ Métodos no informados o ciegos:

- Exploración exhaustiva del espacio de búsqueda hasta encontrar una solución
- La búsqueda no incorpora **conocimiento del dominio**

### ■ Métodos informados o heurísticos:

#### • Búsqueda sistemática

- Exploración de los caminos más *prometedores*
- Se incorpora algo de **conocimiento del dominio** → Funciones Heurísticas que son “pistas” para dirigir y acotar el proceso de búsqueda (eficiencia)

#### • Búsqueda no sistemática (o local): importa la solución y no el camino

- Escalada (hill-climbing) simple y máxima pendiente
- Enfriamiento simulado (simulated annealing)
- Haz local
- Genéticos
- Colonias de hormigas

- **Búsqueda sistemática:** Se encuentran soluciones a un problema generando una secuencia de estados de manera sistemática → **familia primero el mejor**
- **Búsqueda no sistemática (o local):**
  - Encuentran el **mejor estado posible** en base a una función objetivo → problemas de optimización
  - El camino hacia la solución del problema es irrelevante y lo que importa es la solución final → Usan poca memoria (no se guarda la secuencia de estados)

- En la búsqueda local se empieza con una configuración inicial (aleatoria o no) y se hacen cambios (operadores) hasta alcanzar un estado sin sucesores mejores que él.
  - La **función heurística** que evalúa la **calidad de la solución** pero no está necesariamente ligada al coste del camino
  - La función heurística se usará para podar el espacio de búsqueda
- Inconvenientes
  - Bucles y **óptimos locales**
  - No se hace una exploración completa del espacio de estados
  - El **óptimo global** puede no ser alcanzado
  - No se guarda el camino recorrido (en general)
- Ventajas
  - Gasto de memoria es mínimo → no backtracking → algunos tipos de algoritmos de búsqueda local guardan la pista de k nodos

- **Problemas de optimización:** el objetivo es maximizar o minimizar una cantidad, sujeta a una serie de limitaciones que restringen la decisión
  - Matemáticas aplicadas → cálculo diferencial básico
  - Investigación operativa: modelos de optimización y modelos de predicción → problema del viajante TSP
    - Programación lineal (si la  $f$  a optimizar es lineal) o no lineal
      - Técnicas de ramificación y poda
      - Programación dinámica (divide y vencerás y componer)
  - Técnicas heurísticas → aproximadas, basadas en el conocimiento y experiencia en la resolución del problema.
    - Se usan cuando los anteriores no son efectivos (complejidad)
      - Estadística, métodos híbridos, IA, inspiración biológica

- Estado inicial: no está claramente definido (puede estarlo)
- Operadores:
  - Se puede definir cierta noción de nodo “sucesor” o “vecino”
  - En algunos casos, gran cantidad de vecinos
  - Introducimos cierta componente heurística y aleatoria, generando cada vez un único nodo “nodo vecino”
- Estados finales y soluciones:
  - Todos los estados son posibles soluciones, pero se trata de encontrar una solución “buena” (cuantificada por su función de valoración)
  - Si es posible, la mejor
  - Se busca el estado con un óptimo valor (máximo o mínimo)
  - No buscamos la secuencia de operadores (los estados contienen toda la información)

# Búsqueda local.

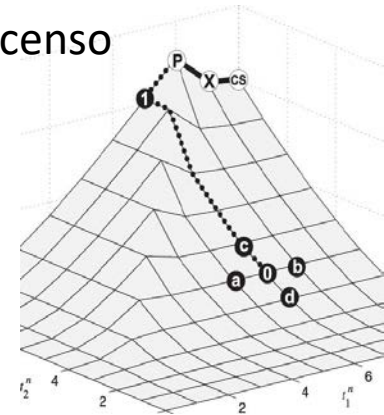
## Estrategia de mejora iterativa

- Empezamos con un estado (solución) inicial (generada aleatoriamente o no)
  - Aplicar a la solución actual una transformación para mejorar la solución
  - La solución mejorada se convierte en la solución actual (solo se mantiene el estado actual)
  - Repetir lo anterior hasta que ninguna acción mejore a la solución actual
- 
- Son irrevocables: sin vuelta atrás
  - Ni óptimos ni completos (pueden quedarse bloqueados en un óptimo local)
  - Permiten heurísticas no admisibles
  - **Escalada**: búsqueda dirigida al máximo siempre
  - **Enfriamiento simulado**: permiten un empeoramiento
  - **Genéticos**: cruces/mutaciones/selección

# Escalada simple (hill climbing)

- Se guarda un único estado
- La función de evaluación alcanza su valor (máximo) *mínimo* en la solución óptima  $f(n) = h'$
- Se mueve siempre en la dirección que *mejora* la función de evaluación → Si no puede, falla
- Se generan los sucesores y en cada paso el nodo actual se sustituye por el **primer** sucesor mejor
- Se usa solo si se tiene una buena  $h'$  y ningún otro conocimiento útil

Ascenso/descenso



evaluar el estado INICIAL

**si** (INICIAL = ESTADO\_OBJETIVO) devolverlo y parar

**si no**

ACTUAL := INICIAL

**mientras** (haya operadores aplicables a ACTUAL y  
no se haya encontrado solución) hacer

seleccionar un operador no aplicado todavía a ACTUAL

aplicar operador y generar NUEVO\_ESTADO

evaluar NUEVO\_ESTADO

**si** NUEVO\_ESTADO = ESTADO\_OBJETIVO

devolverlo y parar

**si no**

**si**  $h'(\text{NUEVO\_ESTADO}) < h'(\text{ACTUAL})$

ACTUAL := NUEVO\_ESTADO



- Escalada simple es muy dependiente tanto de  $h'$  como del orden de generación de hijos
  - No es completo ni óptimo
  - Complejidad en espacio:  $K$  (bytes para almacenar estado)
  - Complejidad en tiempo:  $O(r^p)$
  - Funciona como primero en profundidad guiado por  $h'$ , pero sólo desciende si mejora y no hay vuelta atrás
- Escalada por máxima pendiente
  - En cada paso elijo el hijo con mejor  $h'$  : el nodo actual se sustituye por el sucesor mejor → camino de máxima pendiente)
- Progresa muy rápido hacia una solución, pero suele atascarse en óptimos locales

# Escalada por máxima pendiente. Algoritmo

```
evaluar el estado INICIAL
si INICIAL = OBJETIVO
    devolverlo y parar
si no
    ACTUAL := INICIAL
    mientras no encontrada solución hacer
        SIG := nodo peor que cualquier sucesor de ACTUAL
        para cada operador aplicable a ACTUAL
            aplicar operador y generar NUEVO_ESTADO
            evaluar NUEVO_ESTADO
            si NUEVO_ESTADO = OBJETIVO
                devolverlo y parar
            si no
                si  $h'(\text{NUEVO\_ESTADO}) < h'(\text{SIG})$ 
                    SIG := NUEVO_ESTADO
        si  $h'(\text{SIG}) < h'(\text{ACTUAL})$ 
            ACTUAL := SIG
        si no
            parar
```

Genero todos  
los hijos y me  
quedo con el  
mejor



El problema de la mochila con  
escalada por máxima pendiente

8€/5Kg  
7€/6Kg  
12€/10Kg  
6€/4Kg  
2€/1Kg  
3€/1Kg

16Kg  
Sol Inicial

$h(n)=8$   
8€/5Kg

$h(n)=7$   
7€/6Kg

$h(n)=12$   
12€/10Kg

...

8€/5Kg  
7€/6Kg  
~~12€/10Kg~~  
6€/4Kg  
2€/1Kg  
3€/1Kg

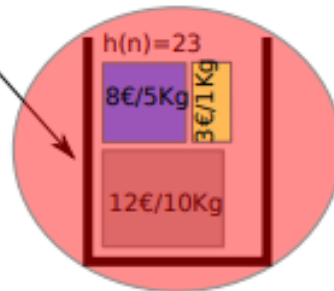
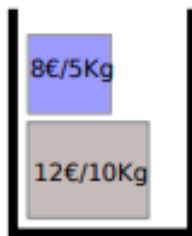
12€/10Kg

$h(n)=20$   
8€/5Kg  
12€/10Kg

$h(n)=19$   
7€/6Kg  
12€/10Kg

$h(n)=18$   
6€/4Kg  
12€/10Kg

...



Sol Final



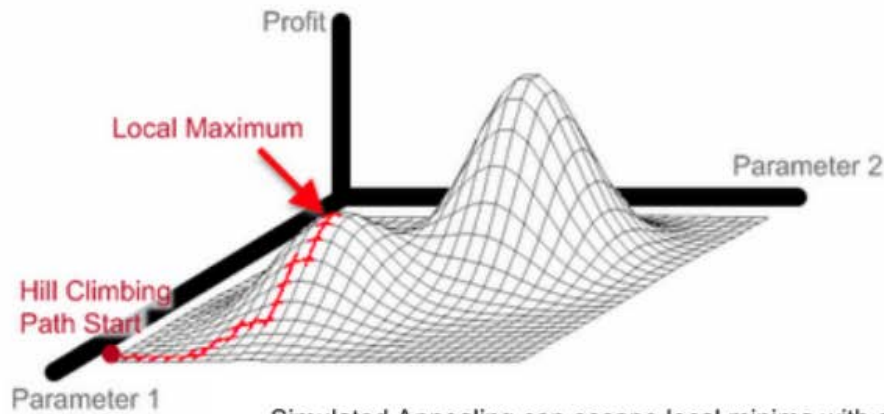
- No son completos → Pueden no encontrar una solución (se paran en un estado que no es objetivo y que no tiene hijos mejores)
- Problemas:
  - **Mínimo local:** Todos los hijos tienen una función heurística peor
  - **Meseta:** Todos los hijos tienen una función heurística igual a la del nodo actual
  - **Cresta:**
    - $h'$  no guía hacia ningún estado objetivo
    - Termina en un mínimo local o en una meseta

- Backtracking a un nodo anterior y seguir el proceso en otra dirección → prohibitivo en espacio → limitar a k nodos → Beam Search
- Reiniciar la búsqueda en otro punto (random restarting Hill climbing)
- Aplicar dos o más operadores antes de decidir el camino a seguir
- Dividir el espacio de búsqueda en regiones y explorar las más prometedoras
- Algoritmos inspirados en analogías físicas y biológicas:
  - **Enfriamiento simulado:** proceso de escalada estocástico inspirado en el proceso de enfriamiento de metales
  - **Algoritmos genéticos:** proceso de escalada en paralelo inspirado en los mecanismos de selección natural

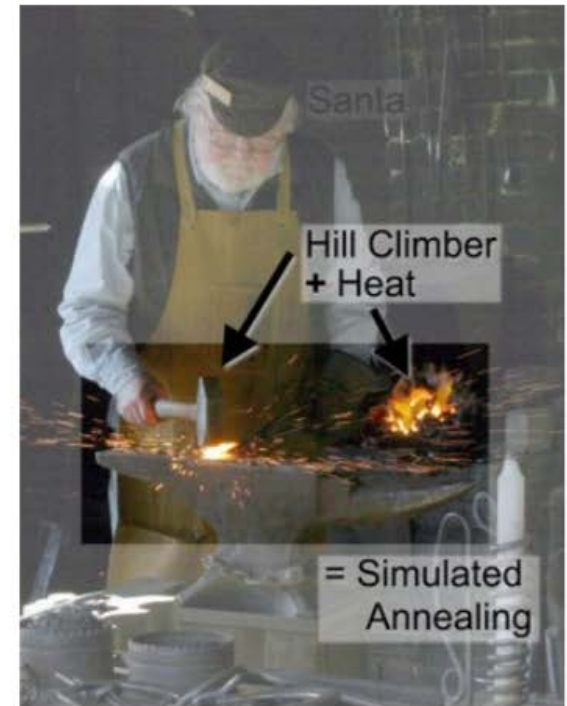
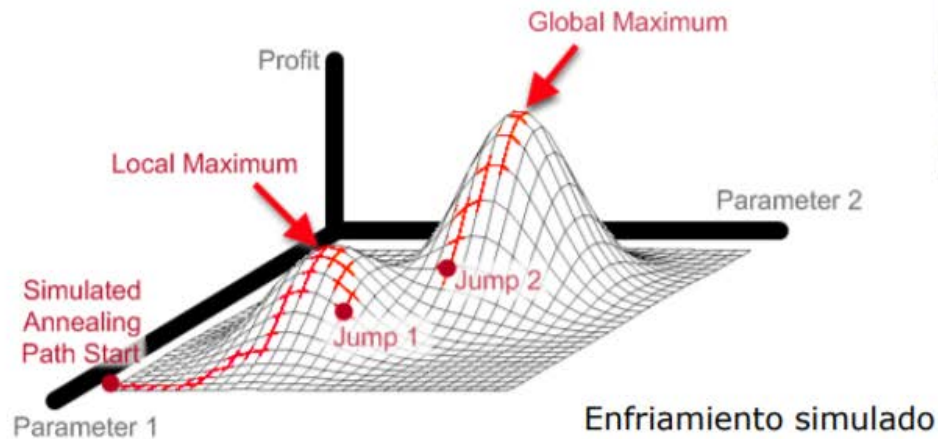
- Guardar sólo un nodo en memoria puede parecer una reacción extrema al problema de limitación de memoria
- El algoritmo de búsqueda por haz local guarda  $k$  nodos
  - Comienza con  $k$  estados generados aleatoriamente
  - En cada paso se generan todos los sucesores de los  $k$  estados
  - Se comprueba si alguno es un objetivo
  - Si no, se seleccionan los  $k$  mejores sucesores y se repite el proceso
- Parecido al proceso de selección natural: los sucesores (descendientes) de un estado (progenitor) pueblan la siguiente generación según su valor (adaptabilidad, salud...)

# Enfriamiento simulado

The problem with hill climbing is that it gets stuck on "local-maxima"

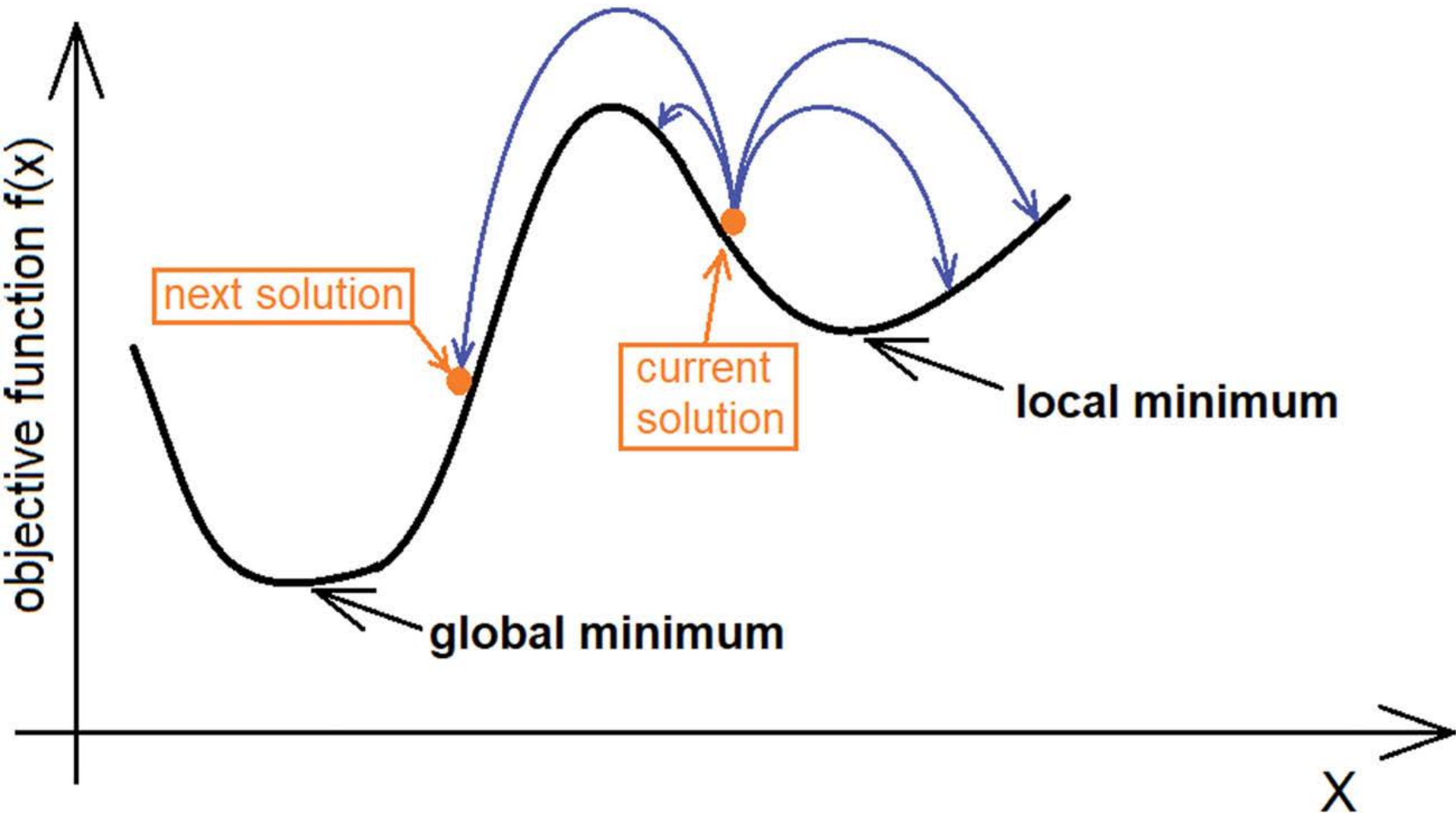


Simulated Annealing can escape local minima with chaotic jumps





# Igual para minimizar



Application of simulated annealing algorithm for 3D coordinate transformation problem solution

Waldemar Odziemczyk

From the journal [Open Geosciences](#) 2020

<https://doi.org/10.1515/geo-2020-0038>

- En metalurgia se sigue un proceso para templar metales y cristales en el que se calientan a una temperatura alta y luego se van enfriando gradualmente para que el material se solidifique en un estado cristalino de energía baja.
- Introduce un componente aleatorio al método de escalada → Se escoge **aleatoriamente** entre los hijos que mejoran la función de evaluación
- También permite moverse en direcciones que empeoran la función de evaluación → Esto nos permite salir de óptimos locales
- La probabilidad con que se permiten movimientos que empeoran disminuye con el empeoramiento y el tiempo
- Con el número de iteraciones disminuye la probabilidad de aceptar soluciones peores que la actual.

- Idea principal:
  - Aceptar probabilísticamente estados “peores”
  - La probabilidad de que un estado peor sea aceptado varía en función del incremento producido en la función de valoración
  - Permitimos así salir de óptimos locales, sin salir del óptimo global
- Algoritmo inspirado en el proceso físico-químico de *enfriamiento de metales*

- Mantiene un conjunto de soluciones candidatas
- La intuición es que a mayor temperatura (al principio), mayor probabilidad de aceptación de soluciones peores.
  - Fase de exploración: se aceptan soluciones mucho peores que la actual al principio de la ejecución
  - Fase de explotación: se aceptan pocas soluciones peores
- Una vez finalizada la iteración, es decir, tras generar  $L(T)$  soluciones vecinas, se enfría la temperatura y se pasa a la siguiente iteración

$$T \leftarrow \alpha (T)$$

- Variable Temperatura,  $T$ , determina en qué medida pueden ser aceptadas soluciones vecinas peores que la actual ( $T$  representa la iteración)
  - Se inicializa a un valor alto,  $T_0$ , y se va reduciendo cada iteración mediante un mecanismo de enfriamiento de la temperatura,  $\alpha(\cdot)$ , hasta alcanzar una Temperatura final,  $T_f$
- En cada iteración se genera un número concreto de vecinos,  $L(T)$ , que puede ser fijo para toda la ejecución o depender de la iteración concreta
- Para cada vecino se aplica el **criterio de aceptación** para ver si sustituye a la solución actual
  - El criterio de aceptación de soluciones vecinas varía en la ejecución del algoritmo. Si la solución vecina **es mejor** que la actual **se acepta**
  - Si es peor, aún **existe la probabilidad** de que el vecino sustituya a la solución actual. Esto permite al algoritmo salir de óptimos locales, en los que la BL clásica quedaría atrapada
  - Esta probabilidad depende de la diferencia de costes entre la solución actual y la vecina,  $\delta$ , y de la temperatura  $T$  (mas temperatura mas probabilidad):

A menor diferencia de costes ( $\delta$ ), mayor probabilidad de aceptación de soluciones peores (mas probabilidad si el salto es pequeño)

$$P_{\text{aceptación}} = \exp(-\delta/T)$$

**Algoritmo:** enfriamiento simulado

**Input:** temperatura inicial y final ( $T_0, T_f$ ); Velocidad de enfriamiento ( $L$ ),  $\alpha$  (mecanismo de enfriamiento), función de calidad  $f$  (maximizar)

$T \leftarrow T_0$

$S_{act} \leftarrow$  Solución inicial (*aleatorio*)

**while**  $T \geq T_f$  **do**

*// paseo aleatorio por el espacio de soluciones*

**for**  $cont \leftarrow 1$  to  $L(T)$  **do**

$S_{cand} \leftarrow$  Selecciona\_sucesor( $S_{act}$ ) (*aleatorio*)

$\delta \leftarrow f(S_{act}) - f(S_{cand})$

**si**  $\delta < 0$  **then** *//si la solución candidata es mejor*

$S_{act} \leftarrow S_{cand}$

**else** con probabilidad  $e^{(-\delta/T)}$

$S_{act} \leftarrow S_{cand}$

$T \leftarrow \alpha(T)$  *//enfriamos*

Se devuelve como solución la mejor  $S_{act}$  visitada

**Algoritmo:** enfriamiento simulado

**Input:** temperatura inicial y final ( $T_0, T_f$ ); **Velocidad de enfriamiento ( $L$ )**,  **$\alpha$  (mecanismo de enfriamiento)**, función de calidad  **$f$**  (maximizar)

$T \leftarrow T_0$

$S_{act} \leftarrow$  Solución inicial (*aleatorio*)

**while**  $T \geq T_f$  **do**

*// paseo aleatorio por el espacio de soluciones*

**for**  $cont \leftarrow 1$  to  **$L(T)$**  **do**      $L(T)$  es el número de vecinos que considero

$S_{cand} \leftarrow$  Selecciona\_sucesor( $S_{act}$ ) (*aleatorio*)

$\delta \leftarrow f(S_{act}) - f(S_{cand})$

**si**  $\delta < 0$  **then**     *//si la solución candidata es mejor*

$S_{act} \leftarrow S_{cand}$

**else** con probabilidad  $e^{(-\delta/T)}$

$S_{act} \leftarrow S_{cand}$

$T \leftarrow \alpha(T)$      *//enfriamos*

Se devuelve como solución la mejor  $S_{act}$  visitada

- Enfriamiento basado en sucesivas temperaturas descendentes fijadas por el usuario
- Enfriamiento con descenso constante de temperatura
- Descenso exponencial:  $T_{k+1} = \alpha \cdot T_k$ 
  - $k = n^{\circ}$  iteración actual,  $\alpha$  constante cercana a 1 (usualmente,  $\alpha \in [0.8, 0.99]$ )
- Criterio de Boltzmann:  $T_k = T_0 / (1 + \log(k))$
- Esquema de Cauchy:  $T_k = T_0 / (1 + k)$
- Para controlar el número de iteraciones (Cauchy modificado):  $T_{k+1} = T_k / (1 + \beta \cdot T_k)$
- Para ejecutar exactamente  $M$  iteraciones  $\Rightarrow$   
 $\beta = (T_0 - T_f) / (M \cdot T_0 \cdot T_f)$



```
while T >= Tf do  
    // paseo aleatorio por el espacio de soluciones  
    for cont ← 1 to L(T) do  
        ¿Cuántos vecinos genero?    Puede ser fijo o variar durante el proceso
```

- L(T) debe ser suficientemente grande como para que el sistema llegue a alcanzar su estado estacionario para esa temperatura
- Un valor fijo o variante que permite decidir mejor cuando finalizar la iteración actual y enfriar
  - Enfriar cuando se dé una de las dos situaciones siguientes:
    - Se ha generado un número máximo de vecinos (máx\_vecinos).
    - Se han aceptado un número máximo de vecinos (máx\_éxitos).
      - $\text{max\_vecinos} > \text{max\_éxitos}$
      - Una buena proporción puede ser  $\text{máx\_éxitos} = 0.1 * \text{máx\_vecinos}$

```
while  $T \geq T_f$  do  
  for  $\text{cont} \leftarrow 1$  to  $L(T)$  do
```

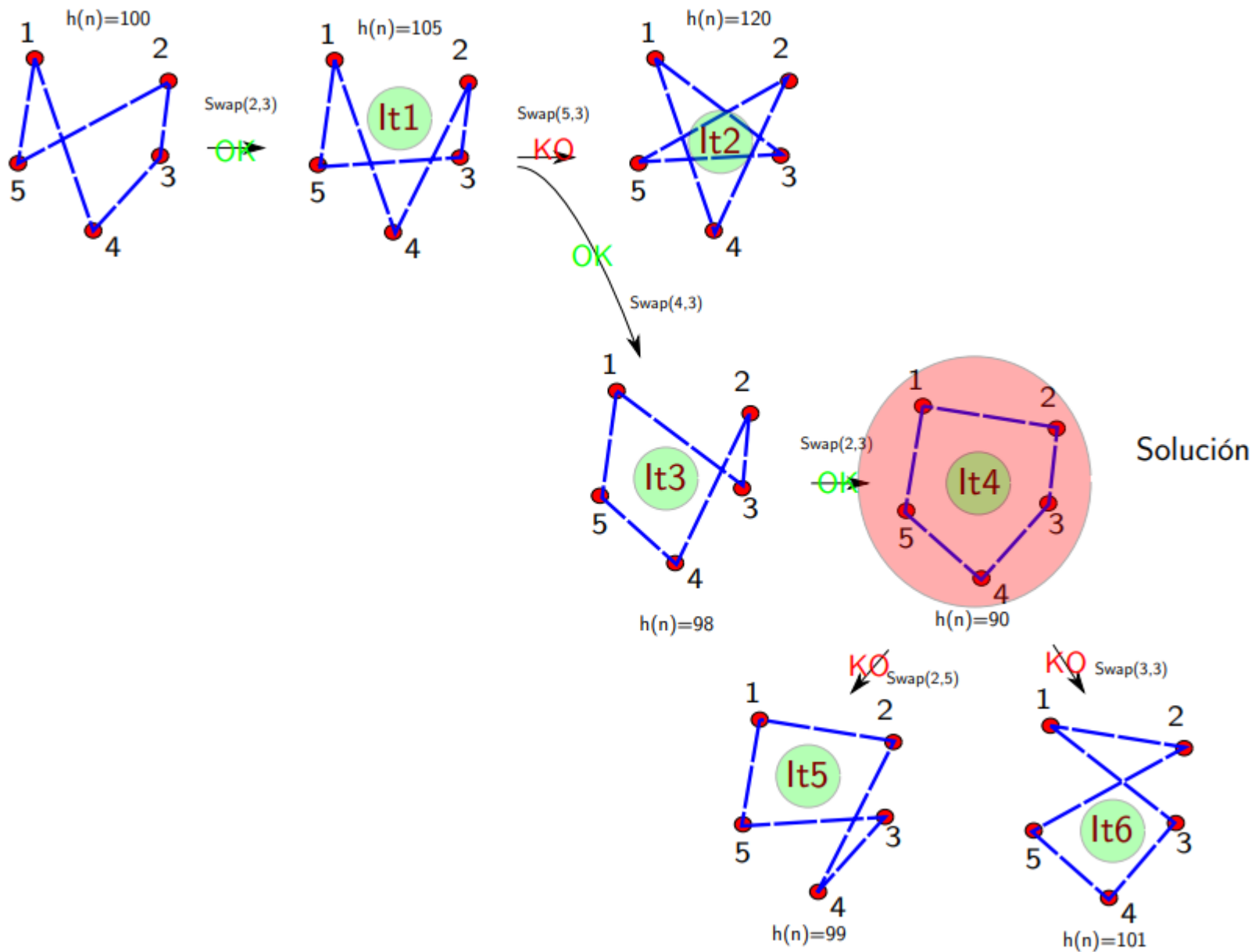
- Cuando  $T$  llega a un valor final  $T_f$  (si lo sabemos)
- Después de un número fijo de iteraciones.
- Cuando no se haya aceptado ningún vecino de los  $L(T)$  generados en la iteración actual ( $\text{num\_éxitos}=0$ )
  - En ese caso, es muy probable que el algoritmo se haya estancado y no vaya a mejorar la solución obtenida
  - Combinando este criterio de parada y la condición de enfriamiento de los  $\text{máx\_vecinos}$  y  $\text{máx\_éxitos}$  se obtiene un equilibrio en la búsqueda que evita malgastar recursos

- Configuración del algoritmo
- Representación:
  - Lista de ciudades en el orden de recorrido.
  - Espacio de soluciones  $N!$
- Generación de vecinos
  - Transformar una solución: inversión de dos ciudades, translaciones, intercambios
- Experimentar con distintas  $T_0$  y  $T_f$  y mecanismos de enfriamiento para controlar el número de iteraciones
- $F$  = Función a optimizar basada en la suma de distancias entre ciudades

$$E = \sum_{i=1}^n \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} + \sqrt{(x_N - x_1)^2 + (y_N - y_1)^2}$$

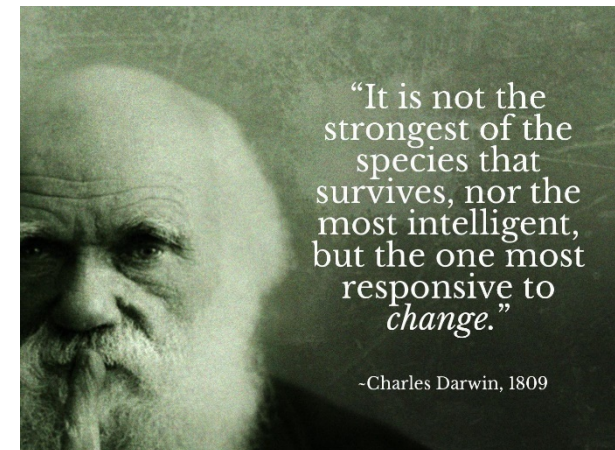
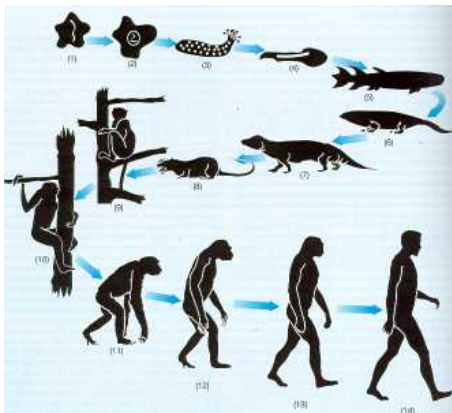
- Valor inicial  $T_0 = 100$
- Velocidad de enfriamiento  $L(T) = 5$  (5 vecinos)
- Enfriamiento: cada iteración la  $T$  se reduce en 10%
  - Descenso geométrico:  $T_k = 0.9 T_{k-1}$
- Criterio de aceptación calculamos la diferencia con la solución actual: función de distancia

# Ejemplo (minimizar distancia entre ciudades)



- Son un tipo de algoritmos dentro de la familia de algoritmos evolutivos.
- Variante de la búsqueda en haz en la que se combinan dos estados padre
- Utilizan el principio de selección natural para resolver problemas de optimización “complicados”
- Se hace evolucionar una población de individuos (cada uno de los cuales representa una posible solución)
- La población se somete a acciones aleatorias semejantes a las de la evolución biológica (mutaciones y recombinaciones genéticas)
- Los individuos se seleccionan de acuerdo con una función de adaptación en función de la cual se decide qué individuos sobreviven (los más adaptados) y cuáles son descartados (los menos aptos)

- Un algoritmo genético es un método de resolución de **problemas de búsqueda y optimización** que imita la teoría de la evolución biológica de Darwin para la resolución de problemas
- Se parte de una **población inicial de soluciones candidatas** de la cual se seleccionan los individuos **más adaptados o capacitados** para luego **reproducirlos y mutarlos** para finalmente obtener la siguiente generación de individuos que estarán más adaptados que la anterior generación

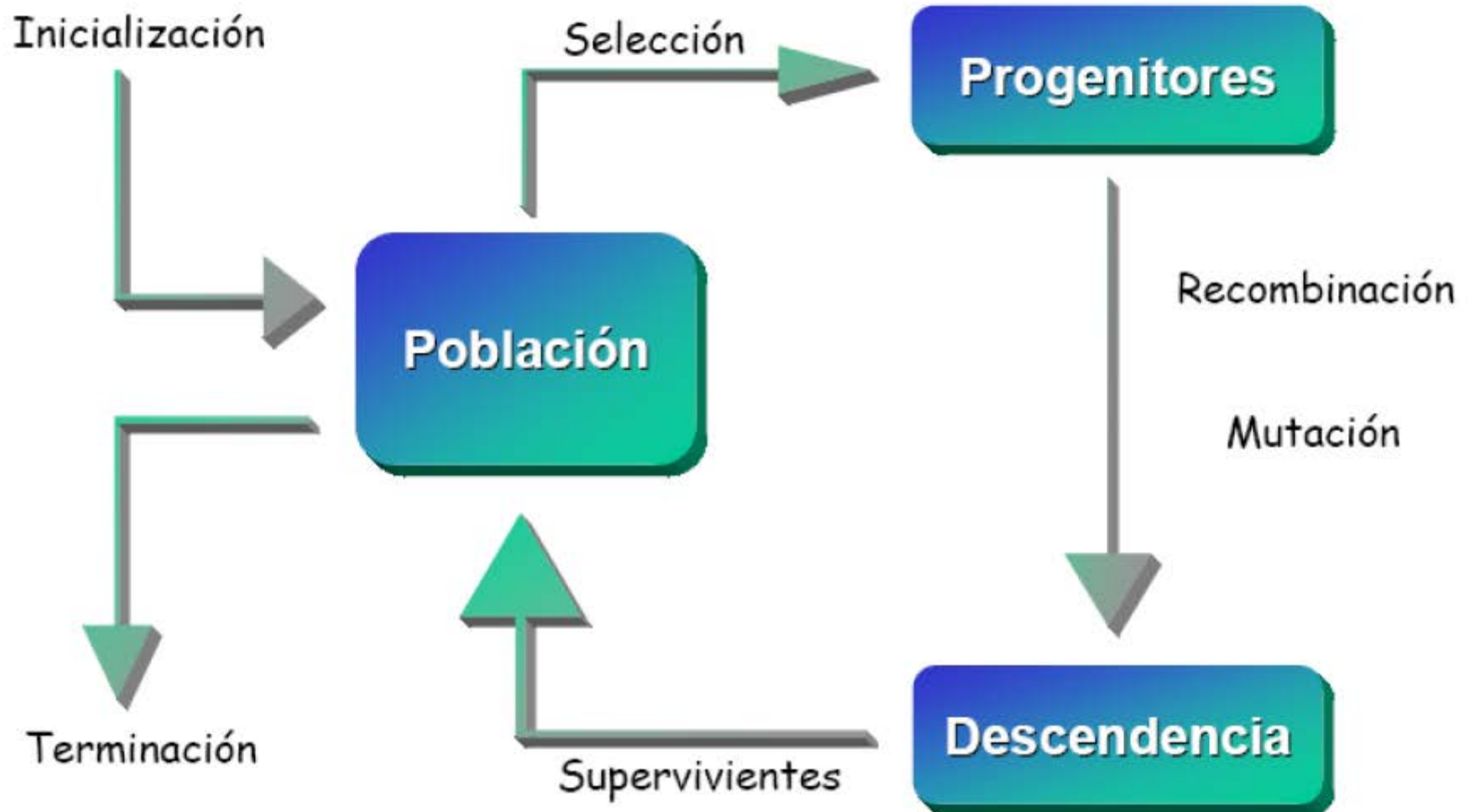


Generar (aleatoriamente) una **población inicial de N** individuos

1. Calcular la valoración o **aptitud** de cada individuo (función de **fitness**)
  - La aptitud indica la capacidad de cada individuo para resolver el problema
2. **Seleccionar** dos individuos (en base a la aptitud)
3. Aplicar operador genético de **cruce (crossover)** → Habrá que tener en cuenta la **probabilidad de cruce**
4. Aplicar operador genético de **mutación** → Habrá que tener en cuenta la **probabilidad de mutación**

Ciclar 1-4 hasta obtener una población con **M individuos**



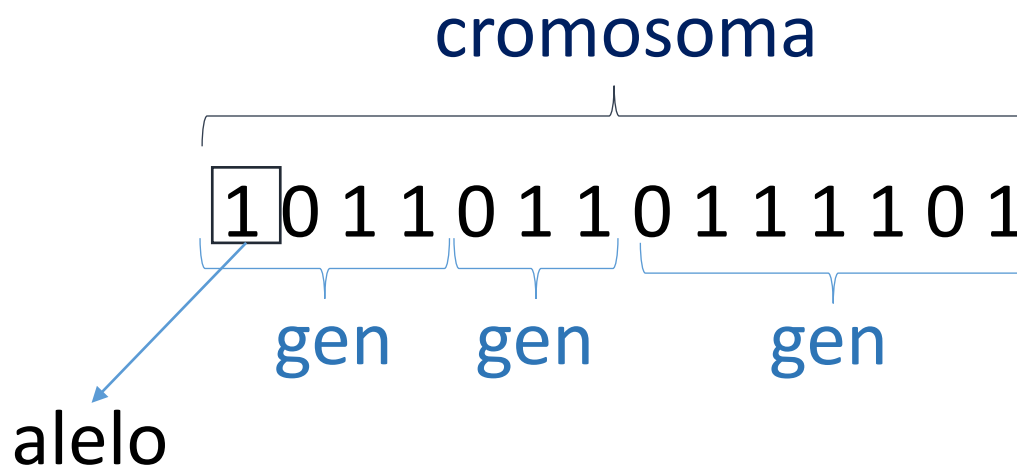


- **Representación** de las soluciones potenciales del problema
- Mecanismo para crear la **población inicial** de posibles soluciones → generalmente se hace de forma aleatoria
- **Función de evaluación** que clasifique las soluciones según su aptitud
- **Operadores genéticos** (selección, cruce y mutación)
  - No se definen sucesores como en búsqueda exhaustiva
- Valores para los **parámetros** propios del algoritmo
  - Tamaño de la población inicial
  - Probabilidad de cruce, probabilidad de mutación
  - Terminación: número máximo de generaciones o tamaño  $M$  de la población final

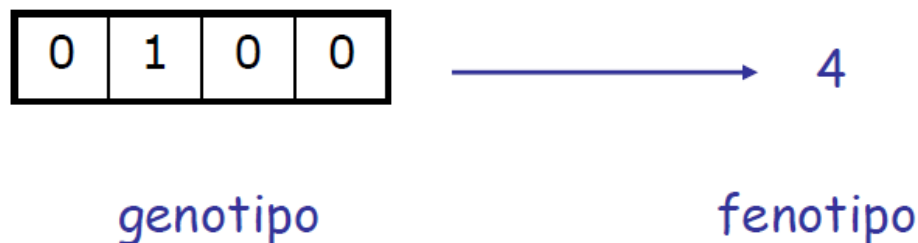
- **Cromosoma:** solución potencial a un problema
  - Formado por **genes**
- Los **genes** pueden codificarse con un valor entero, real... aunque una forma típica de codificar cada gen es con valores binarios
  - Se asigna un determinado número de bits a cada gen
  - El número de bits asignados dependerá del grado de ajuste que se desee alcanzar → **No todos los genes tienen porque estar codificados con el mismo número de bits**
- **Alelo:** Cada uno de los bits pertenecientes a un gen

# Algoritmos genéticos.

## Representación (II)

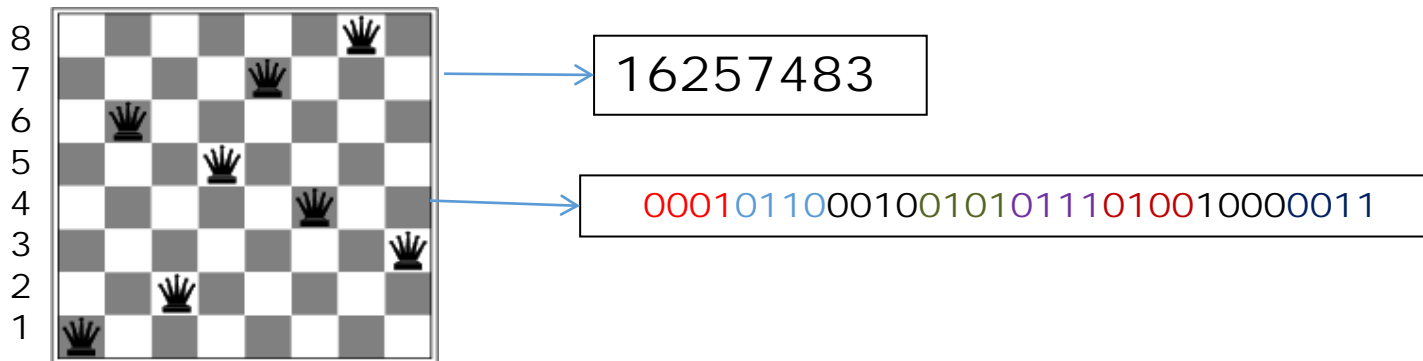


fenotipo: es la decodificación del cromosoma: el valor obtenido al pasar de la representación binaria a la usada por la función objetivo



# Problema de las 8 reinas. Representación

- Población : Combinaciones de 8 reinas en un tablero
  - Cada gen es una reina
  - Cada cromosoma o individuo es una combinación de 8 reinas
- Posibles codificaciones de *cromosoma*
  - Lista de 8 dígitos con valor de 1 al 8
    - Su posición en la lista es la columna
    - Su valor es la fila
  - Cadena de 1's y 0's . Total =  $8 * 4 = 32$  bits
  - Se puede hacer con valores del 0 al 7  $\rightarrow$  3 bits por gen



# 1. Población inicial

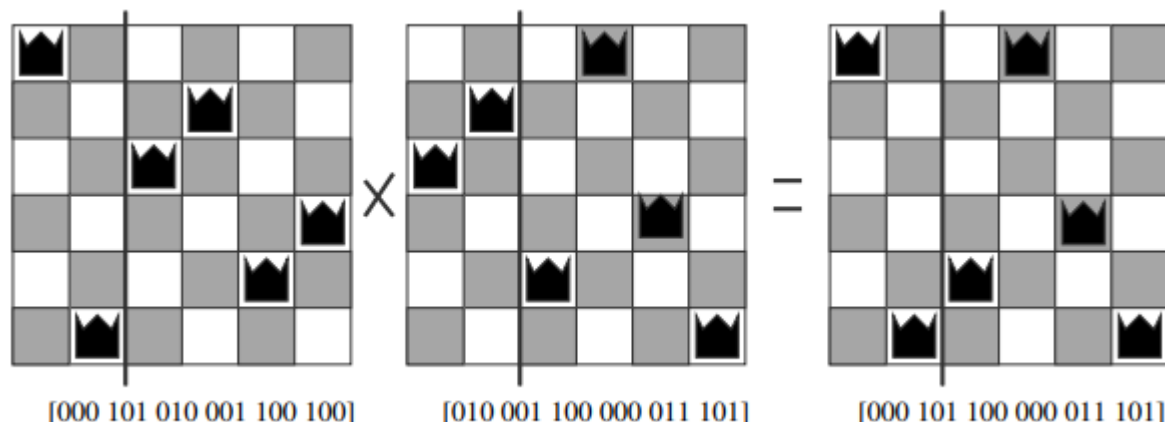
- La inicialización de la población suele ser aleatoria pero conviene asegurar una mezcla adecuada de todos los alelos posibles → Incluir en la población inicial valores distintos para todos los genes
- Todos los individuos de la población inicial deben de ser válidos (rango valores)

## 2. Función de aptitud (o fitness). Características

- Evalúa la bondad o aptitud de los individuos de la población en relación al problema planteado
- Debe penalizar malas soluciones y premiar las buenas
- Debe tomar siempre valores positivos
  - Cuidado con errores (por ejemplo, división por cero)
- Debe mantener una cierta regularidad → individuos que codifiquen valores próximos en el espacio de búsqueda deben de poseer valores de fitness similares
- Se calcula muchas veces así que debe ser un cálculo rápido.
  - Un cálculo lento del valor aptitud puede afectar negativamente al algoritmo y hacerlo muy lento

### 3. Operadores genéticos

- No hay función genera\_sucesores
- Los operadores genéticos nos permiten generar nuevas soluciones candidatas
  - Selección  $\rightarrow$  aridad 1
  - Cruce  $\rightarrow$  aridad 2
  - Mutación  $\rightarrow$  aridad 1



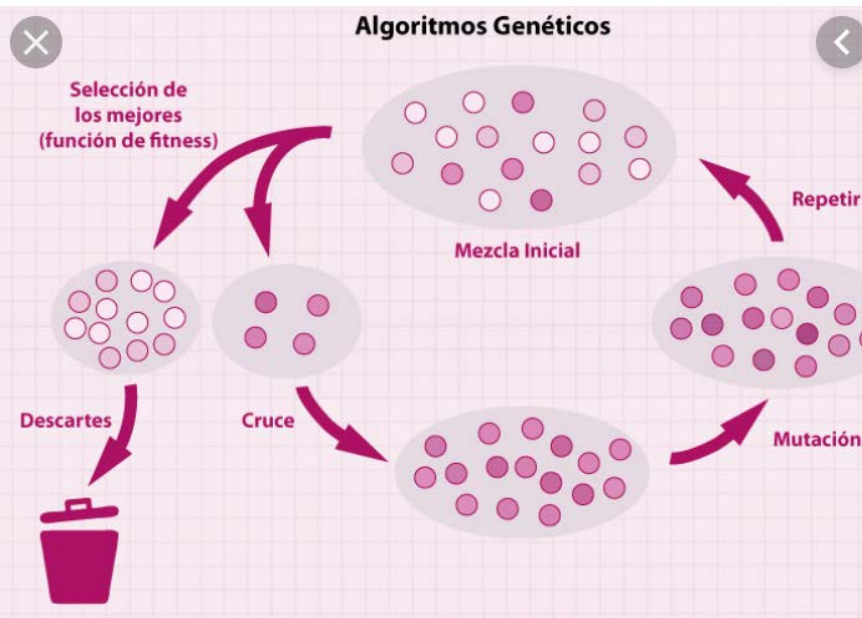


- Encargado de **escoger qué individuos se van a reproducir**
- La selección de un individuo estará relacionada con su valor de aptitud → mayor número de oportunidades de reproducción a los individuos más aptos
  - Más copias de los individuos mejor adaptados.
- Pero no se deben eliminar por completo las opciones de reproducción de los individuos menos aptos → si lo hacemos la población se volvería homogénea
- Existen muchas técnicas diferentes para seleccionar a los individuos para la siguiente generación (incluso selección aleatoria)
- Vamos a ver los más básicos y se pueden usar de forma exclusiva o combinada
  - Selección **elitista**
  - Selección **por ruleta**
  - Selección **por torneo**

# Selección elitista

Se seleccionan de dos en dos para pasarlos al cruce

Garantiza la selección de los miembros más aptos de cada generación



En general no funciona bien el **elitismo puro**, sino una forma modificada por la que el individuo mejor, o algunos de los mejores, son copiados hacia la siguiente generación

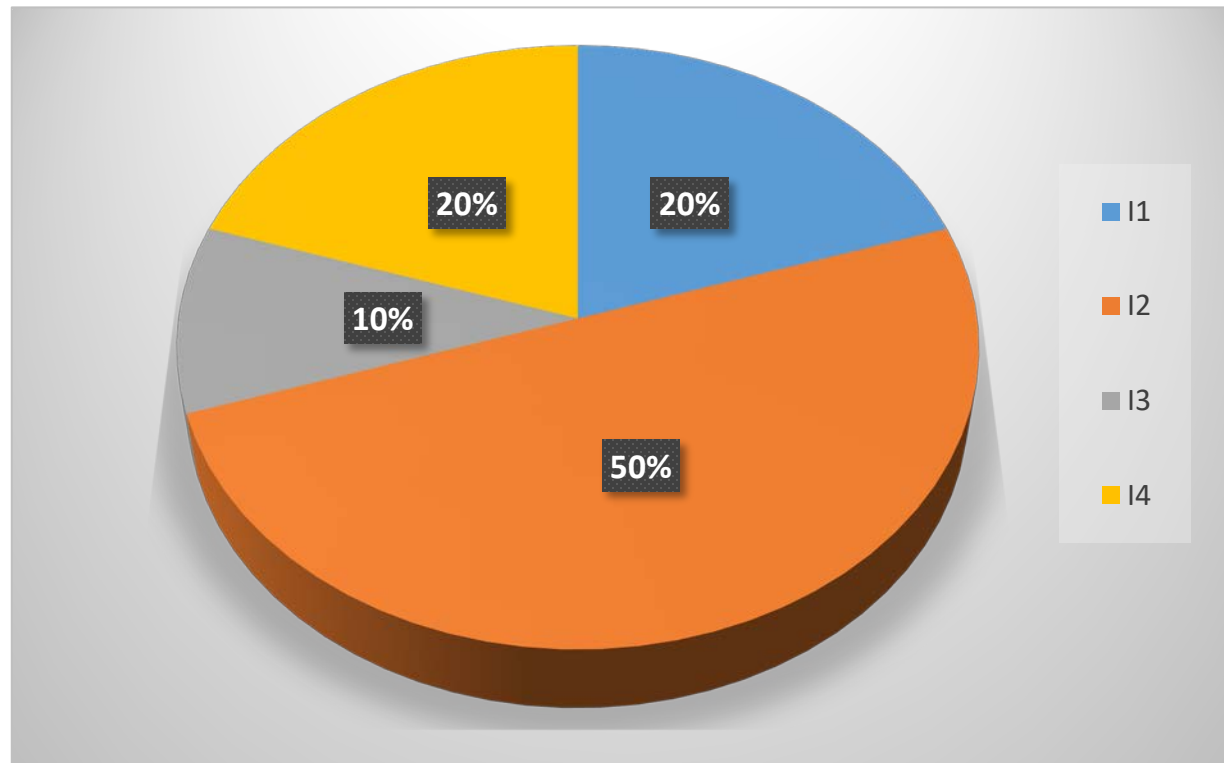
# Selección por ruleta (o Montecarlo)

- Cada uno de los individuos de la población tiene una probabilidad de ser escogido (la suma de todos será la unidad)
- Los mejores individuos (según función de evaluación) recibirán una probabilidad (porción de la ruleta) mayor que la recibida por los peores
- Desventajas:
  - Ineficiente a medida que aumenta el tamaño de la población ( $O(n^2)$  )
  - El peor individuo puede ser seleccionado más de una vez



# Selección por ruleta. Ejemplo

Individuo	Función de evaluación
I1	2
I2	5
I3	1
I4	2



# Selección por ruleta

- Para cada  $i$  se calcula la probabilidad de ser escogido

$$p_i = \text{fitness}_i / \text{fitness}_{\text{TOTAL}}$$

- Y se calcula la **probabilidad acumulada** a cada individuo  $i$ :

$$q_i := p_1 + \dots + p_i$$

- Para seleccionar un individuo basta con generar un número aleatorio ( $a$ ) del intervalo  $[0..1]$  y se selecciona el individuo  $i$  que cumpla:  $q_{i-1} < a \leq q_i$
- Ejemplo:

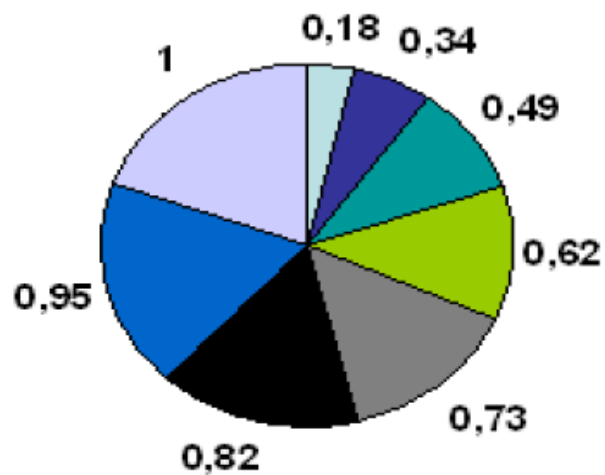
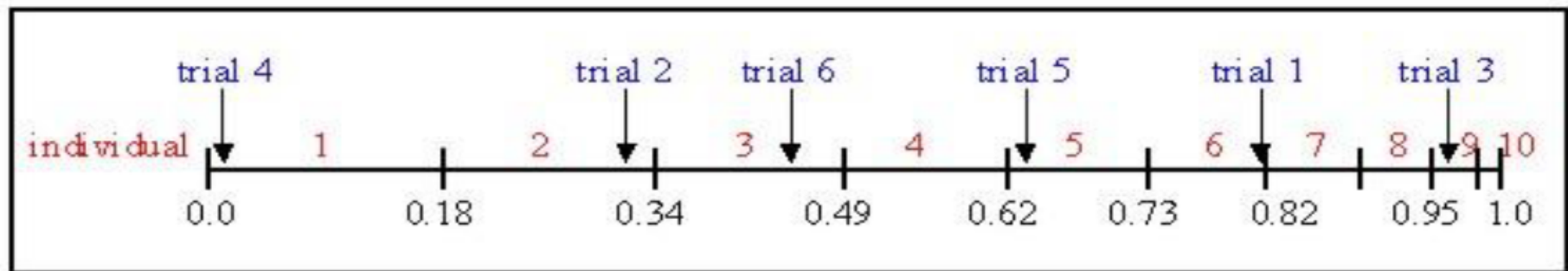
Individuo	Función de evaluación	Pi	Qi
I1	2	0,2	0,2
I2	5	0,5	0,7
I3	1	0,1	0,8
I4	2	0,2	1

←  $a=0,13$  (pointing to Qi=0,2)

←  $a=0,75$  (pointing to Qi=0,8)

$$\text{fitness}_{\text{TOTAL}} = 10$$

individuo	1	2	3	4	5	6	7	8	9	10
fitness	2.0	1.8	1.6	1.4	1.2	1.0	0.8	0.6	0.4	0.2
Prob. selección	0.18	0.16	0.15	0.13	0.11	0.09	0.07	0.06	0.03	0.02



Para seleccionar 6 individuos el proceso se repite 6 veces: 0.81, 0.32, 0.96, 0.01, 0.65, 0.42.



- La idea principal es realizar la selección en base a comparaciones directas entre individuos
- Se eligen **subgrupos de tamaño  $p$  de individuos** de la población
- $p$  = presión de selección
- Los miembros de cada subgrupo compiten entre ellos y se elige a **un** individuo de cada subgrupo para el cruce
- Este método tiene un cierto grado de elitismo → si siempre se elige el mejor de cada subgrupo
- Elitismo global:  $p$ = tamaño de la población
  - Torneo en el que participan todos los individuos de la población

## ■ **Determinista:**

- Se selecciona al azar un número  $p$  de individuos
- Es común el tamaño de subgrupo  $p = 2$
- De entre los  $p$  individuos seleccionados se selecciona el más apto para cruce (se selecciona un individuo por torneo)
- Un individuo puede participar mas de una vez

## ■ **Probabilística:** En vez de escoger siempre el mejor se genera un número aleatorio del intervalo $[0..1]$ , si es mayor que un parámetro $p$ (fijado para todo el proceso evolutivo) se escoge el individuo más apto y en caso contrario el menos apto

➤ Generalmente  $0.5 < p \leq 1$



# Selección por torneo. Presión de selección

- Variando el número  $P$  de individuos que participan en cada torneo se puede modificar la presión de selección
  - $P$  alto: muchos individuos en cada torneo → presión de selección elevada → los peores individuos tienen pocas oportunidades
  - $P$  bajo: tamaño del torneo reducido → la presión de selección disminuye → los peores individuos tienen más oportunidades
- Si se opta por un método con una **alta presión de selección** se centra la búsqueda de las soluciones en un entorno próximo a las mejores soluciones actuales
- Si se opta por una presión de selección menor se deja el camino abierto para la **exploración** de nuevas regiones del espacio de búsqueda

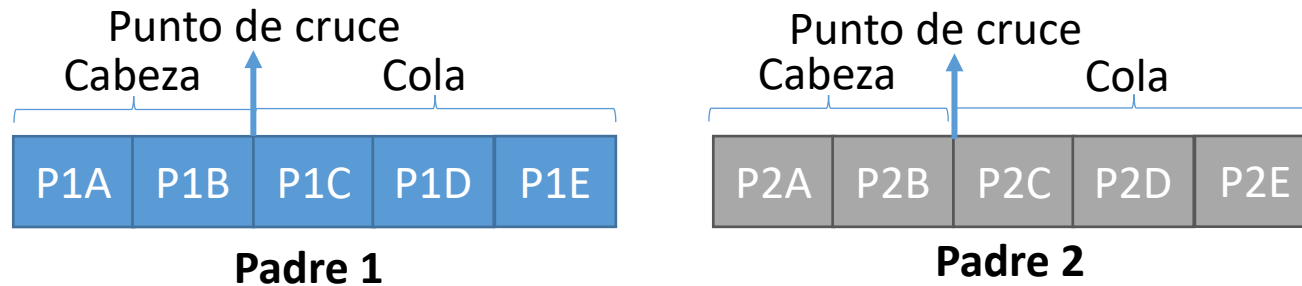
- Existen muchas posibilidades para algoritmos de selección:
  - Métodos que buscan mejorar la eficiencia computacional
  - Métodos que buscan mejorar el número de veces que los mejores o peores individuos pueden ser seleccionados
  - ...
  - Muestreo determinístico
  - Escalamiento sigma
  - Selección por jerarquías
  - Estado uniforme
  - Sobrante estocástico
  - Brecha generacional, etc.

- Una vez seleccionados los individuos, éstos son recombinados para producir la descendencia que se insertará en la siguiente generación
- **Intuición:** dados dos individuos correctamente adaptados al medio, si se obtiene una descendencia que comparta genes de ambos, existe la posibilidad de que los genes heredados sean precisamente los causantes de la bondad de los padres → Al compartir las características buenas de dos individuos, la descendencia, o al menos parte de ella, debería tener una bondad mayor que cada uno de los padres por separado
- **Probabilidad de cruce** Es la probabilidad de que dos cromosomas intercambien sus genes
  - Se suele usar un valor en torno a 0,7 (70%)
- Los algoritmos más empleados **son cruce de 1 punto** o **cruce de 2 puntos**

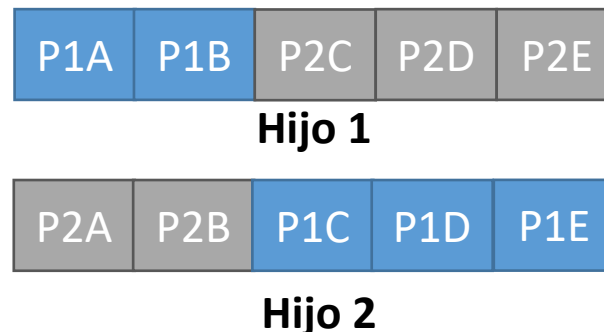
- **Destructiva:** Los descendientes se insertarán en la siguiente generación aunque sus padres tengan mejor ajuste
  - Los genes de los padres continuarán en la población → en posteriores cruces se podrán volver a obtener estos padres, recuperando la bondad previamente perdida
- **No destructiva:** La descendencia pasará a la siguiente generación únicamente si supera la bondad del ajuste de los padres
  - Con este tipo de estrategia garantizamos que pasen a la siguiente generación los mejores individuos

# Cruce de 1 punto

- Una vez seleccionados dos individuos para cruzar se cortan sus cromosomas por **un punto seleccionado aleatoriamente** para generar dos segmentos diferenciados en cada uno de ellos: la cabeza y la cola



- Se **intercambian las colas** entre los dos individuos para generar los nuevos descendientes



# Ejemplo cruce de un punto

Antes del cruce

**0011|011010**  
**1110|010001**

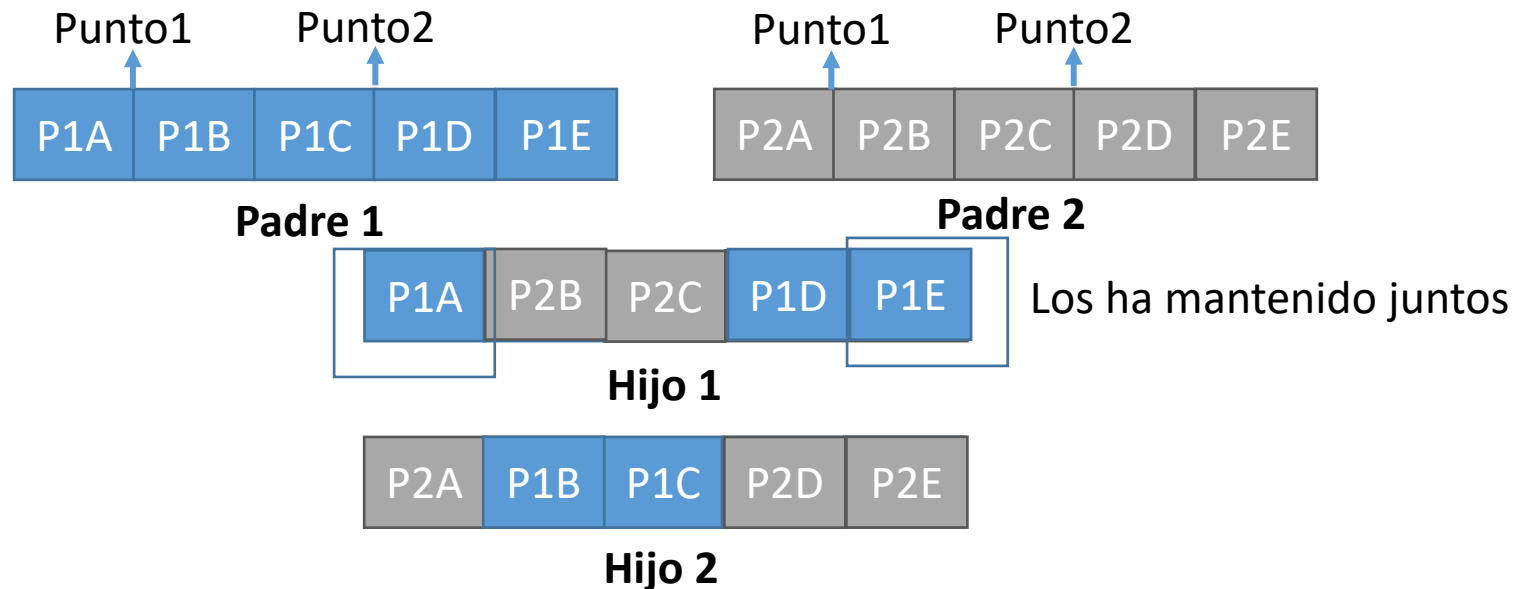


Después del cruce

**0011010001**  
**1110011010**

# Cruce de 2 puntos

- El cruce de 1 punto depende mucho del orden en que aparecen los genes
  - Es más probable que sigan juntos genes que estén más cerca
  - Nunca mantiene juntos genes de extremos opuestos
- En el cruce de 2 puntos se hacen dos cortes
- Para generar la descendencia se escoge el segmento central de uno de los padres y los segmentos laterales del otro



# Ejemplo de cruce de dos puntos

Antes del cruce

**001 | 101 | 1010**  
**111 | 001 | 0001**



Después del cruce

**0010011010**  
**1111010001**



- En la naturaleza en ocasiones se producen errores en la replicación de material genético → Un hijo puede tener material genético que no proviene de ninguno de sus padres
- La mutación puede tener distintas consecuencias:
  - Catastróficas → El hijo no es viable y muere
  - Neutrales → La mutación no influye en la calidad del hijo
  - Ventajosas → La mutación mejora la calidad del hijo
- En la mutación es esencial la aleatoriedad
- En los algoritmos genéticos la mutación introduce **diversidad** en la población



- Consiste en modificar un(os) **gen(es)** del cromosoma con una probabilidad
  - La probabilidad de mutación suele ser muy baja, generalmente menor al 10%
- Se realizan mutaciones para garantizar que ningún punto del espacio de búsqueda tenga una probabilidad nula de ser examinado → exploración

- Mutación después del cruce: mutarlos antes de introducirlos en la nueva población
- O utilizarse de manera conjunta con el operador de cruce:
  - Se seleccionan dos individuos de la población para realizar el cruce
  - Se mutan los descendientes según **probabilidad de mutación**
    - Probabilidad de que un gen de un cromosoma mute
    - Se suele usar un valor muy bajo, en torno al 10% (0,1)
- El operador de cruce puede descubrir áreas prometedoras lejos de los padres
  - Sólo el cruce puede combinar lo mejor de dos padres
  - Sólo la mutación puede introducir nuevos alelos en la población

- **Reemplazo aleatorio:** Se varia aleatoriamente un gen de un cromosoma
  - Si la codificación es binaria se niega un bit (un alelo del gen)
- **Intercambio de alelos:** Se intercambian los valores de dos alelos del cromosoma
- **Incremento/Decremento:** Se incrementa o decrementa un gen en una pequeña cantidad generada aleatoriamente → La codificación no debe ser binaria
- **Multiplicación:** Se multiplica un gen por un valor aleatorio próximo a 1 → La codificación no debe ser binaria
- ...

```
poblacion := crearPoblaciónInicial(N);
repeat
    nuevaPoblacion := vacia;
    getFitness(poblacion);
    for i=1 to N/2 do
        seleccionar(poblacion, x, y);
        randomVal := getRandom();
        if (randomVal < probabilidadCruce)
            cruzar(x,y,child1, child2);
        else
            child1 := x;
            child2 := y;
        end if;
        mutar(child1, probabilidadMutacion);
        mutar(child2, probabilidadMutacion);
        añadir(nuevaPoblacion, child1, child2);
    end for;
    poblacion := nuevaPoblacion;
until solucionEncontrada or tiempoSuperado or
numGeneracionesSuperado;
return mejorIndividuo(poblacion);
```

Si no hay cruce los dos individuos seleccionados pasan sin cruzarse

Mismo tamaño de la población

**Criterio de codificación:** Procedimiento que hace corresponder a cada punto del dominio del problema una cadena (paso del genotipo al fenotipo).

**Criterio de tratamiento de los individuos no factibles:** No siempre existe una correspondencia uno a uno entre el dominio de un problema y el conjunto de las cadenas binarias de tamaño de la codificación (el espacio de búsqueda): no todas las cadenas codifican elementos válidos del dominio del problema. Se necesitan procedimientos para distinguirlas.

**Criterio de inicialización:** Construcción de la población inicial con la que se aplica el bucle básico del AG.

**Criterio de parada:** condiciones en las que se considera que el AG ha llegado a una solución aceptable.

## Fases de la ejecución de un algoritmo genético "ideal":



Generaciones iniciales:  
Distribución aleatoria de la población.



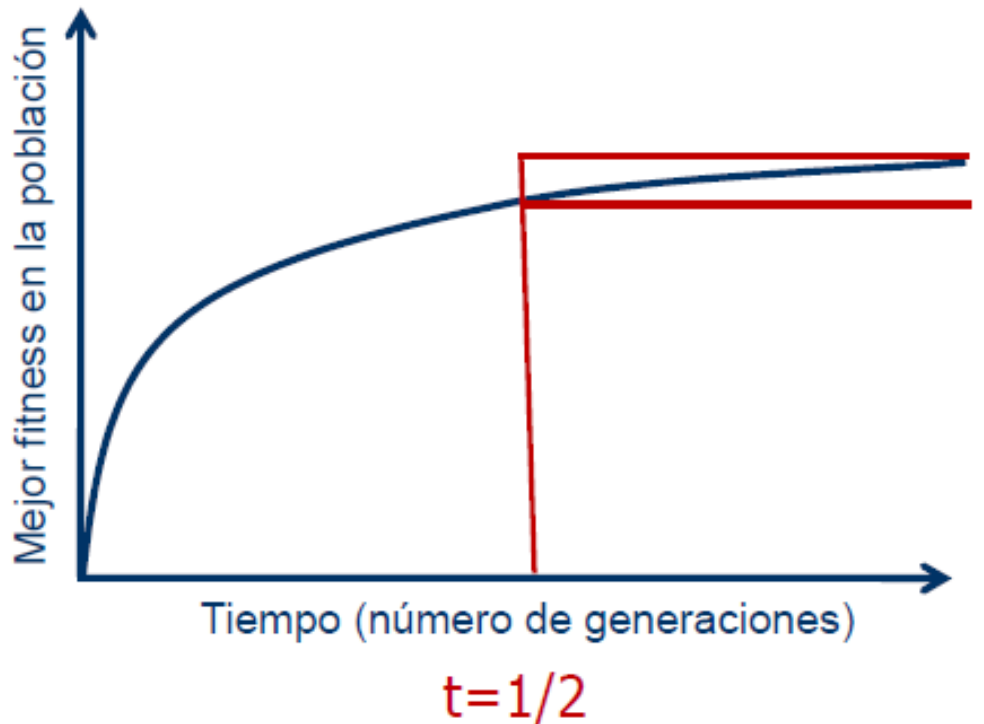
Generaciones intermedias:  
Población cerca de las colinas.



Generaciones finales:  
Población concentrada en las  
cimas de las colinas más altas.

## ¿Merece la pena una ejecución larga?

Depende (a veces, pueden obtenerse mejores resultados con varias ejecuciones más cortas).



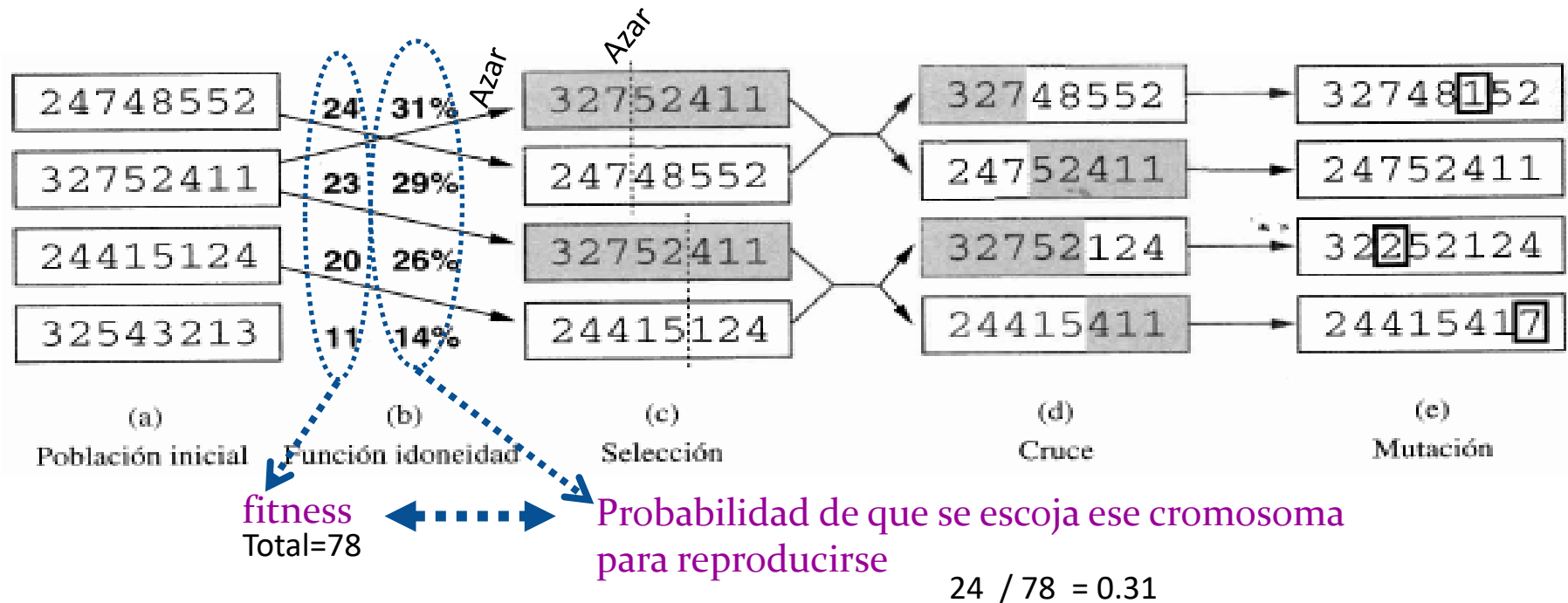
Progreso en  
la segunda  
mitad de la  
ejecución

Progreso en  
la primera  
mitad de la  
ejecución

Puede merecer la pena invertir en inicialización



- Ejemplo de función de idoneidad o fitness: número de parejas de reinas que NO se atacan (max = solución 28)
  - selección ruleta / cruce 1 punto / mutación aleatoria



# Ejemplo sencillo: cuadrados

$$f(x) = x^2$$

valor de  $x$  que hace que la función  $f(x)$  alcance su valor máximo, para  $x$  con valores entre 0 y 31 (enteros)

El máximo se tiene para  $x = 31$  ( $f = 961$ )

**Población inicial** de 6 individuos entre 0 y 31 (5 bits)

(1)	(2)	(3)	(4)	(5)
1	(0,1,1,0,0)	12	144	6
2	(1,0,0,1,0)	18	324	3
3	(0,1,1,1,1)	15	225	2
4	(1,1,0,0,0)	24	576	5
5	(1,1,0,1,0)	26	676	4
6	(0,0,0,0,1)	1	1	1

- (1) = Número que le asignamos al individuo.
- (2) = Individuo en codificación binaria (genotipo)
- (3) = Valor de  $x$  (fenotipo)
- (4) = Valor de  $f(x)$
- (5) = pareja asignada para el torneo

## SELECCIÓN POR TORNEO

En los torneos ganan el 1, el 2 y el 5

Nueva población después de la selección

(1)	(2)	(3)	(4)
1	(0,1,1,0,0)	5	1
2	(0,1,1,0,0)	3	3
3	(1,0,0,1,0)	2	3
4	(1,0,0,1,0)	6	1
5	(1,1,0,1,0)	1	1
6	(1,1,0,1,0)	4	1

## CRUCE

- (3) = pareja para cruce
- (4) = punto de cruce

## POBLACIÓN TRAS EL CRUCE 1X

VALOR  $x$  Y  $F(X)$

(1)	(2)	(3)	(4)
1	(0,1,0,1,0)	10	100
2	(1,1,1,0,0)	28	784
3	(0,1,1,1,0)	14	196
4	(1,0,0,0,0)	16	256
5	(1,1,0,1,0)	26	676
6	(1,0,0,1,0)	18	324

Clasificación de los procesos de optimización según la naturaleza de las soluciones:

- **Numéricas**: Si la solución queda completamente especificada en términos de un conjunto de  $m$  parámetros o atributos.
- **Combinatorias**: Si para especificar la solución hay que especificar un conjunto de  $m$  parámetros y el orden (total o parcial) con que estos se combinan para dar dicha solución.

Optimización de funciones

8 reinas

- Problemas de secuenciación: lo importante es qué elementos aparecen junto a otros.
- Los operadores de cruce y mutación habituales pueden llevar a soluciones inadmisibles
  - No respetan unas restricciones mínimas sobre valores válidos
  - Se puede penalizar con la función de fitness
- La mutación se refiere al cromosoma completo, no se mutan genes individuales

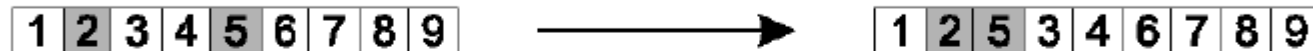
8 reinas



Resultado después de cruce

# Operadores de mutación para permutaciones

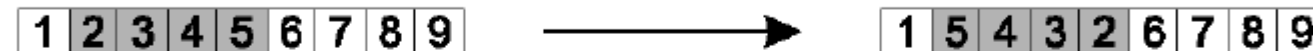
- **Inserción:** Elegir dos alelos aleatoriamente y colocar el segundo justo después del primero.



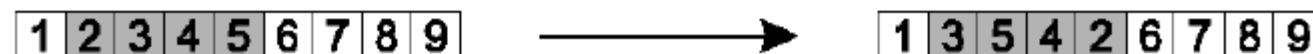
- **Intercambio:** Seleccionar dos alelos aleatoriamente e intercambiarlos.



- **Inversión:** Seleccionar dos alelos aleatoriamente e invertir la cadena entre ellos.



- **Revuelto [scramble]:** Seleccionar un subconjunto de genes y reordenar aleatoriamente los alelos (el subconjunto no tiene por qué ser contiguo).



## Cruce de orden

(se preserva el orden relativo de los elementos)

- Seleccionar una parte arbitraria del primer padre.
- Copiar esta parte en el hijo.
- Copiar los números que no están en la primera parte...
  - ... empezando desde el punto de cruce.
  - ... utilizando el orden en el que aparecen en el 2º padre
  - ... volviendo al principio del cromosoma cuando hayamos llegado al final.

El segundo hijo se crea intercambiando los papeles de los padres.

## Cruce de orden

(se preserva el orden relativo de los elementos)

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

Selección del primer padre



			4	5	6	7		
--	--	--	---	---	---	---	--	--

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---

Los que faltan: 1,2,3,8,9 se copian en el orden del 2º padre empezando desde el 2º punto de cruce

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---

Copia del segundo padre



3	8	2	4	5	6	7	1	9
---	---	---	---	---	---	---	---	---

9	3	7	8	2	6	5	1	4
---	---	---	---	---	---	---	---	---



## PMX [Partially-Mapped Crossover]

A partir de los padres P1 y P2:

1. Elegir un segmento aleatorio y copiar desde P1.
2. Desde el primer punto de cruce, buscar elementos de ese segmento en P2 que no se han copiado.
3. Para cada uno de esos elementos  $i$ , buscar qué elemento  $j$  se ha copiado en su lugar desde P1.
4. Colocar  $i$  en la posición ocupada  $j$  de P2 (sabemos que no estará ahí, ya que lo hemos copiado ya).
5. Si el lugar ocupado por  $j$  en P2 ya se ha rellenado en el hijo ( $k$ ), poner  $i$  en la posición ocupada por  $k$  en P2.
6. El resto de elementos se rellena de P2.



## PMX [Partially-Mapped Crossover]

1 2 3 4 5 6 7 8 9



4 5 6 7

9 3 7 8 2 6 5 1 4

1 2 3 4 5 6 7 8 9



2 4 5 6 7 8

9 3 7 8 2 6 5 1 4

1 2 3 4 5 6 7 8 9

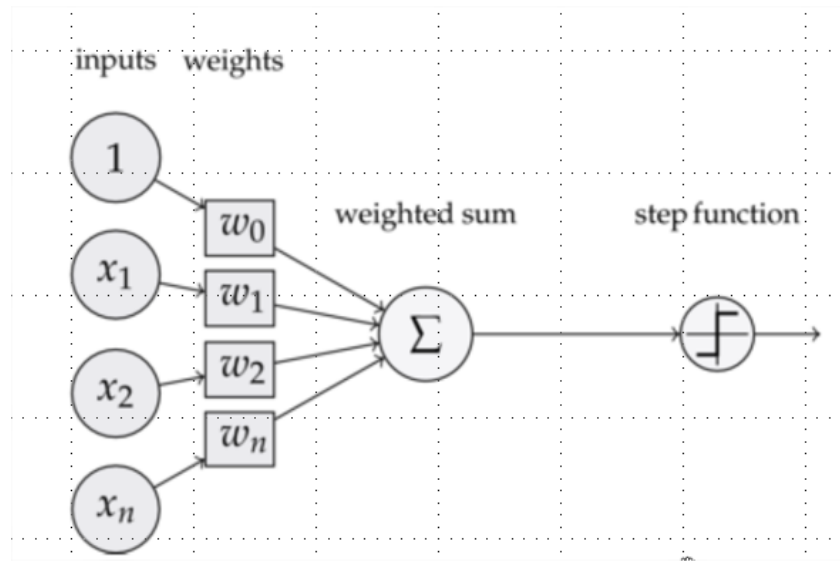


9 3 2 4 5 6 7 1 8

9 3 7 8 2 6 5 1 4



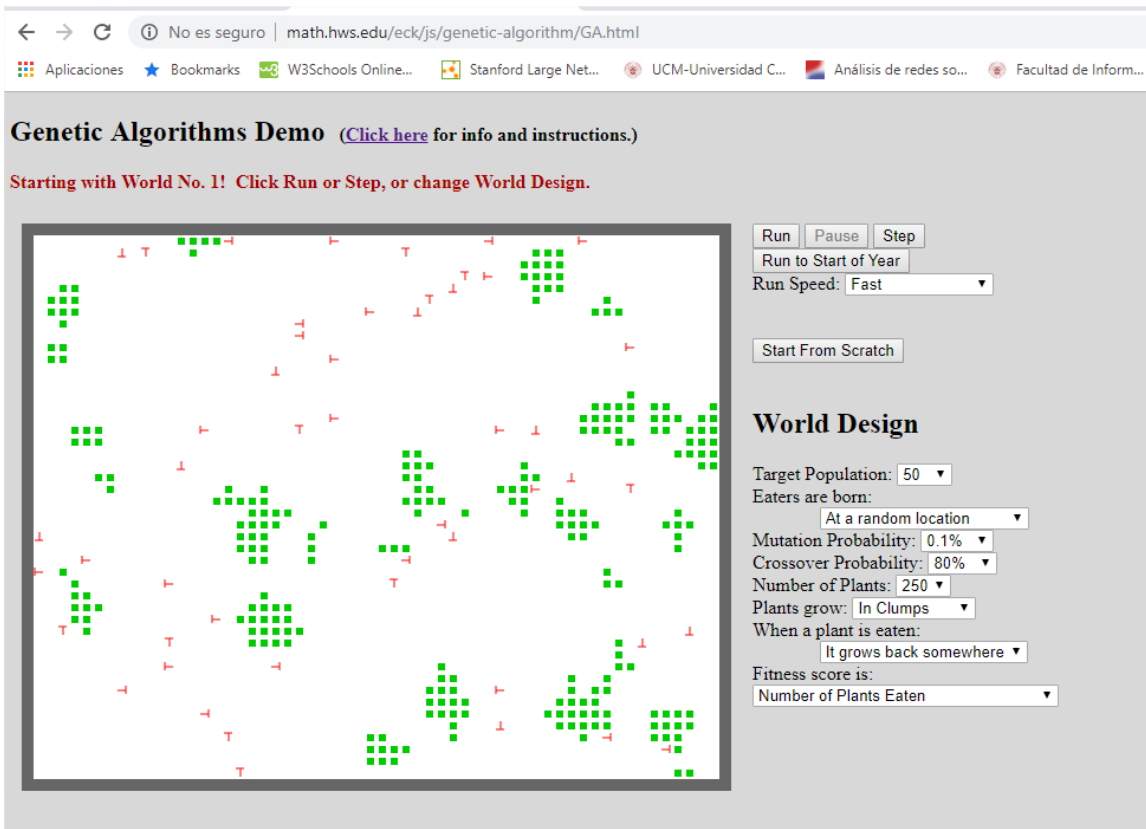
- Optimización (p.e. asignación de horarios)
- Aprendizaje máquina
- Bases de datos (optimización de consultas)
- Reconocimiento de patrones
- Generación de gramáticas
- Planeación de movimientos de robots
- Predicción
- Aprendizaje



The red things are Eaters, which eat the plants (the green things).

A generation or "year" lasts 250 days. As generation follows generation, the Eaters might evolve to become better eaters. The Eaters' behavior changed.

The chromosome consists of 64 rules (one for each combination of one of 16 states and one of 4 items-in-view). Each rule specifies two things: an action and a new state.



The screenshot shows a web browser window with the URL [math.hws.edu/eck/js/genetic-algorithm/GA.html](http://math.hws.edu/eck/js/genetic-algorithm/GA.html). The page title is "Genetic Algorithms Demo" with a link for info and instructions. Below the title, it says "Starting with World No. 1! Click Run or Step, or change World Design." The main area is a 2D grid world simulation. On the left, there's a large square area filled with green squares (plants) and red T-shaped figures (eaters). On the right, there's a control panel with buttons for "Run", "Pause", and "Step". Below these are "Run to Start of Year" and "Run Speed: Fast" (a dropdown menu). There's also a "Start From Scratch" button. Under the heading "World Design", there are several settings: "Target Population: 50", "Eaters are born: At a random location", "Mutation Probability: 0.1%", "Crossover Probability: 80%", "Number of Plants: 250", "Plants grow: In Clumps", "When a plant is eaten: It grows back somewhere", and "Fitness score is: Number of Plants Eaten".

State 0 and 15 "state" of the Eater.

4 actions:

1. Move forward 1 square
2. Move backwards 1 square
3. Turn in place 90° left
4. Turn in place 90° right

Fitness: defined as the number of plants eaten

- Simplicidad conceptual y amplia aplicabilidad
- No garantizan encontrar solución óptima
- Mejores resultados (en media) que técnicas tradicionales de búsqueda en el espacio de estados en muchos problemas del mundo real
- No requieren conocimiento del dominio (aunque mejora el rendimiento si se tiene conocimiento)
- Pueden combinarse con otras técnicas de búsqueda
- Buenos para exploración: capaces de resolver problemas para los cuales no se conoce solución alguna
- Suelen funcionar bien cuando es suficiente una solución subóptima, o varias soluciones válidas, no estado inicial único.

- Técnica de aprendizaje no supervisado (aprendizaje inductivo por observación y descubrimiento)
  - La creación de nuevos ejemplos (puntos de búsqueda) por el algoritmo es una apuesta inductiva
- Método de búsqueda local
  - Los Algoritmos Genéticos (AGs) son métodos estocásticos de búsqueda ciega de soluciones cuasi-óptimas.
  - Población de soluciones potenciales en vez de un solo estado, lo cual los hace menos sensibles a quedar atrapadas en mínimos/máximos locales
  - Los algoritmos evolutivos no necesitan conocimiento específico sobre el problema que intentan resolver
- Método de optimización
  - Búsqueda y optimización son dos facetas de un mismo concepto: Búsqueda es el proceso / optimización el resultado del proceso.

- Algoritmos Evolutivos: teoría y casos prácticos: Lourdes Araujo, Carlos Cervigón (Spanish Edition) Kindle Edition Amazon
- Enfriamiento simulado  
<https://sci2s.ugr.es/sites/default/files/files/Teaching/GraduatesCourses/Algoritmica/Tema03-EnfriamientoSimulado-12-13.pdf>
- **Berzal, F.** Algoritmos geneticos.  
<http://elvex.ugr.es/decsai/iaio/slides/G2%20Genetic%20Algorithms.pdf>
- **Ceccaroni, L.** 2017. Inteligencia Artificial Búsqueda local. [http://www.cs.upc.edu/~luigi/II/IA-2007-fall/2c-Busqueda-local-\(es\).pdf](http://www.cs.upc.edu/~luigi/II/IA-2007-fall/2c-Busqueda-local-(es).pdf)
- **Coello C.:** Introducción a la Computación Evolutiva. CINVESTAV-IPN, 2017.  
<http://delta.cs.cinvestav.mx/~ccoello/compevol/apuntes.pdf>
- **Eiben, A.E. & Smith, J.E.** 2007. Introduction to Evolutionary Computing  
<http://www.cs.vu.nl/~gusz/ecbook/ecbook.html>
- **Meseguer, Pedro.** Búsqueda heurística I. <http://www.iiia.csic.es/~pedro/busqueda1-introduccion.pdf>
- **Pérez, A.** Búsqueda local. <http://delta.cs.cinvestav.mx/~adiaz/anadis/LocalSearch.pdf>