Universidad de Oviedo

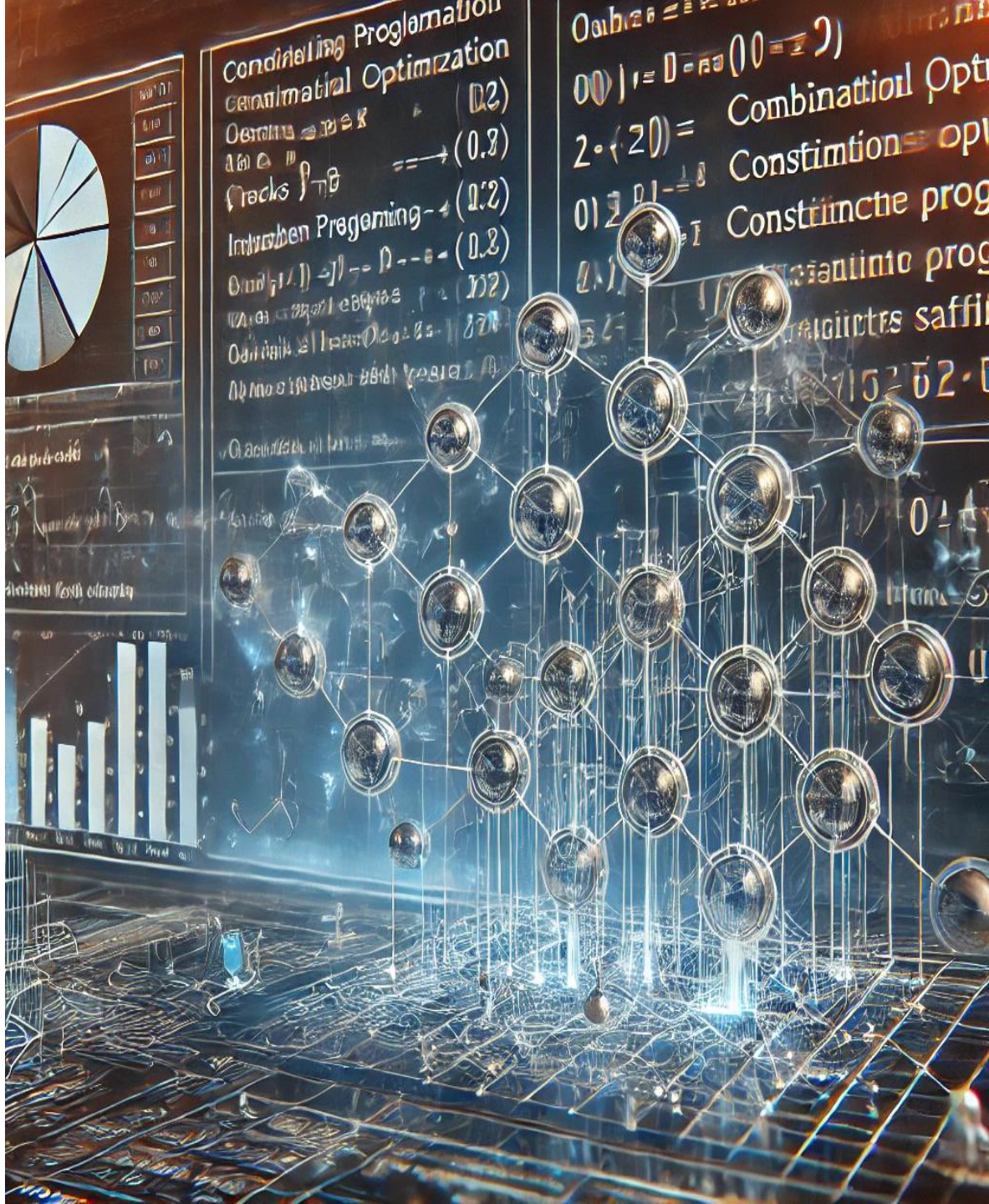
Intelligent Scheduling and Optimization

# Técnicas de Inteligencia Artificial para la Optimización y Programación de Recursos

## Algoritmos evolutivos


Miguel Ángel González Fernández
mig@uniovi.es

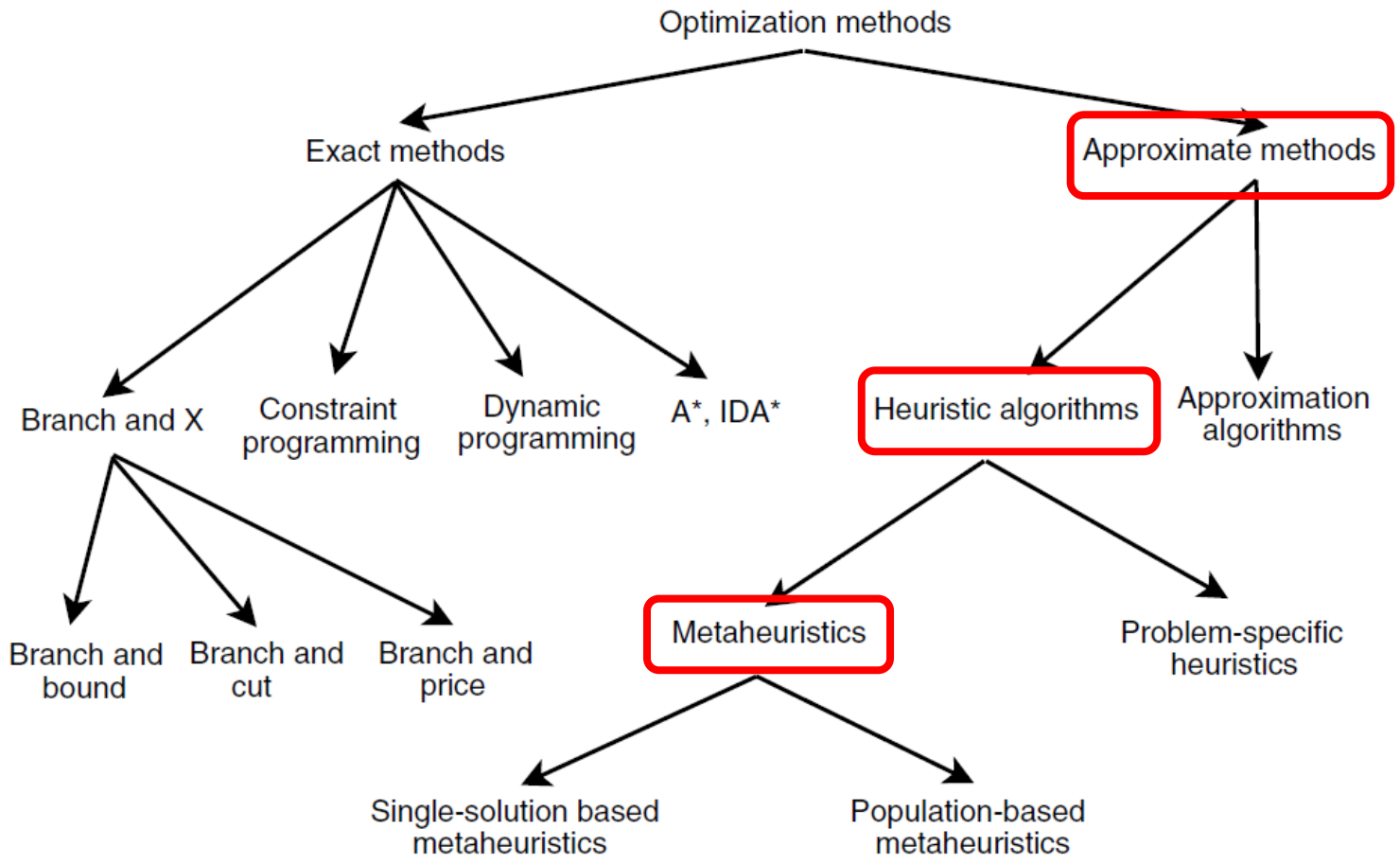
Ciencia de la Computación e Inteligencia Artificial
Departamento de Informática

# Contenidos

1. **Metaheurísticas: introducción y clasificación**
2. Algoritmos genéticos
3. Introducción a la búsqueda local
4. Algoritmos meméticos
5. Introducción a la optimización multiobjetivo
6. Conclusiones

# Metaheurísticas
## Definición



# Metaheurísticas
## Definición

- Los algoritmos metaheurísticos son algoritmos aproximados de optimización y búsqueda de propósito general
- Son procedimientos iterativos que guían una heurística subordinada
- No están ligadas a ningún tipo especial de problema, son métodos generales que se pueden ser modificados para adaptarse a cualquier problema específico
- Es aconsejable su uso si se cumplen los siguientes puntos:
  1. En problemas de gran complejidad computacional (NP-duros)
  2. Los algoritmos exactos (programación dinámica, ramificación y poda, backtracking, A*, …) requieren mucho tiempo de cálculo o son ineficientes o incluso imposibles de aplicar
  3. No necesitamos la solución óptima, nos conformamos con conseguir una buena solución en un tiempo razonable

# Metaheurísticas
## Ventajas

Ventajas de las metaheurísticas:

- Resuelven cualquier tipo de problema general
- Muy flexibles
- Gran éxito en la práctica
- A menudo son robustas respecto al tamaño del problema, la instancia u otras variables aleatorias
- A veces son la única alternativa práctica
- Fáciles de implementar
- Fáciles de paralelizar

# Metaheurísticas
## Inconvenientes

Inconvenientes de las metaheurísticas:

- Son algoritmos aproximados (no exactos)
- La convergencia a la solución óptima puede no estar garantizada
- A menudo necesitan información o técnicas específicas del problema
- No suelen tener una base teórica formal
- Diferentes ejecuciones pueden dar lugar a diferentes soluciones al mismo problema (es decir, suelen ser estocásticas)
- Suelen tener multiples parámetros que hay que ajustar

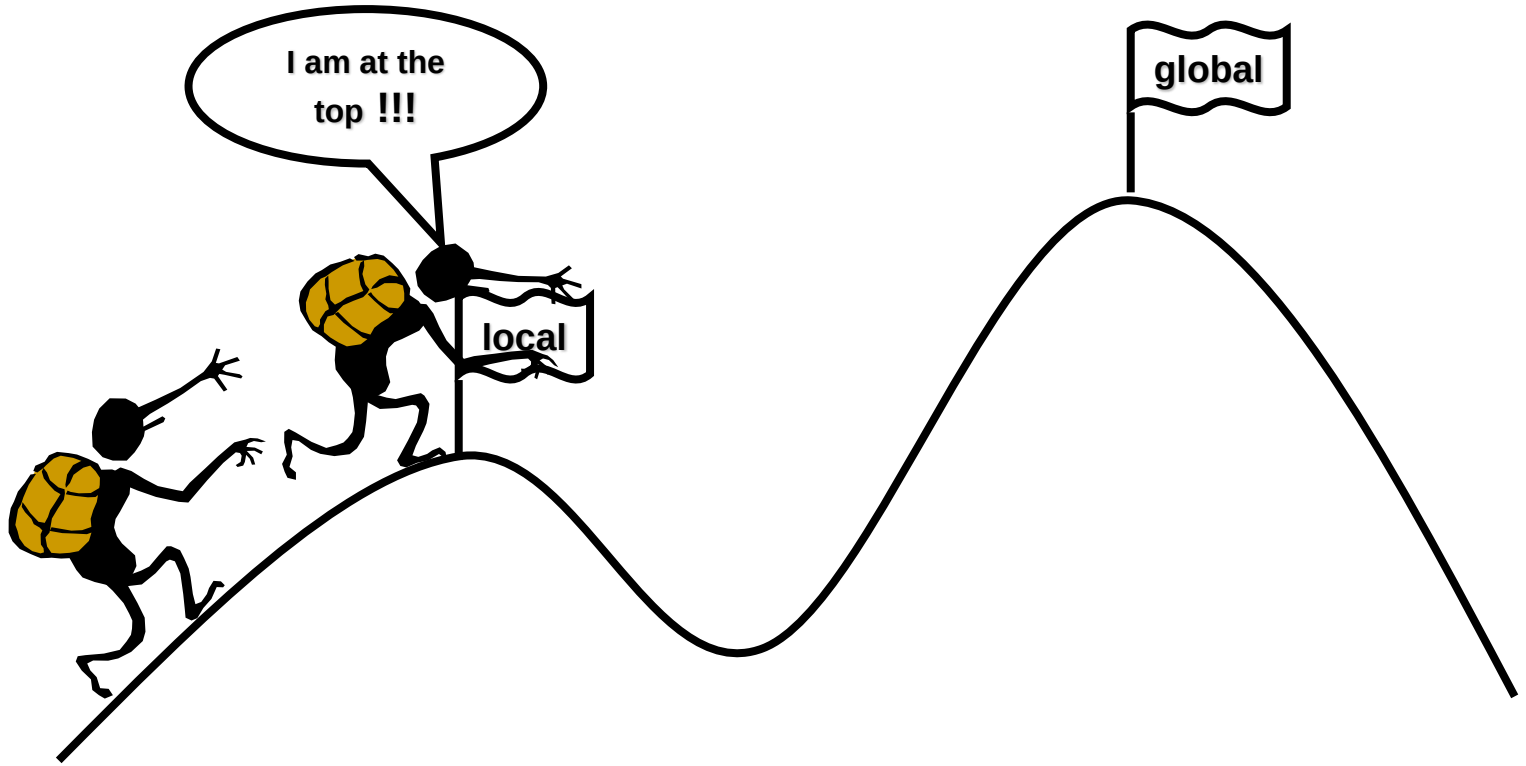
# Metaheurísticas
## Una posible clasificación

- Basada en trayectorias: se parte de una solución inicial y se reemplaza iterativamente por otras de mejor calidad (escalada, enfriamiento simulado, búsqueda tabú, búsqueda local iterada, …)

- Basada en poblaciones: el proceso de búsqueda considera múltiples soluciones en el espacio de búsqueda que van evolucionando (algoritmos genéticos, scatter search, optimización de enjambres de partículas, colonias de hormigas, …)

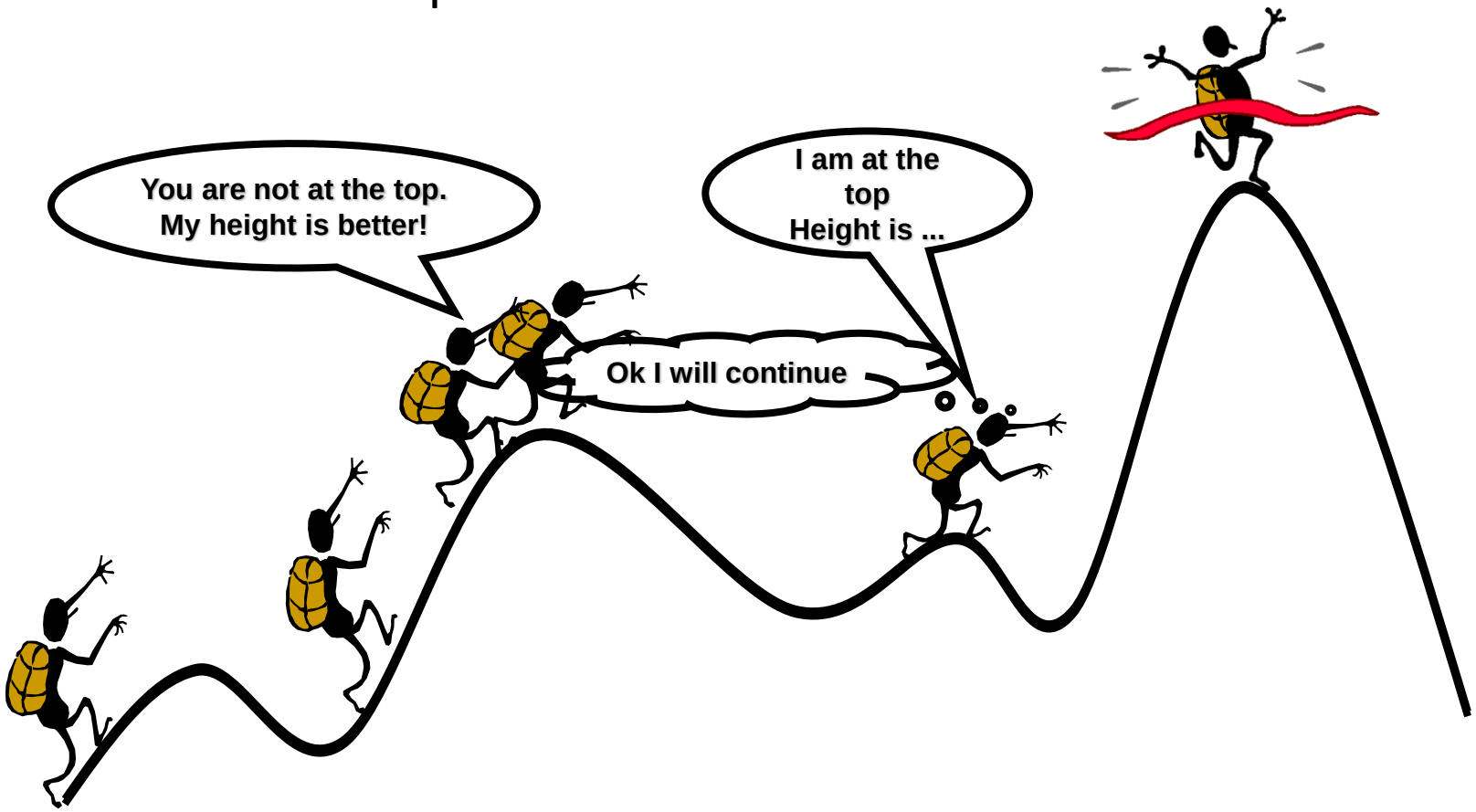
# Metaheurísticas
## Una posible clasificación

Metaheurísticas basadas en trayectorias:



# Metaheurísticas
## Una posible clasificación

Metaheurísticas basadas en poblaciones:



# Metaheurísticas
## Intensificación vs Diversificación

- **Intensificación:** cantidad de esfuerzo dedicado a buscar en la region específica actual (explotación del espacio de búsqueda)

- **Diversificación:** cantidad de esfuerzo dedicado a buscar (explorar) en otras regiones distantes del espacio de búsqueda (exploración del espacio de búsqueda)

Para alcanzar buenas soluciones, cualquier metaheurística debe alcanzar un equilibrio adecuado entre intensificación y diversificación, que son dos características opuestas en el proceso de búsqueda. Dicho equilibrio es necesario para:

- Identificar rápidamente regiones del espacio de búsqueda que son prometedoras y contienen soluciones de alta calidad
- No consumir demasiado tiempo y recursos de búsqueda en regiones del espacio de búsqueda no prometedoras o ya suficientemente exploradas

# Aplicaciones de las metaheurísticas

- Son dominios paradigmáticos la optimización combinatoria y la optimización numérica
- Optimización combinatoria
  - Problema del viajante de comercio (Traveling salesman problem)
  - Problema de la mochila (Knapsack problem)
  - Problema de Asignación Cuadrática (Quadratic Assignment Problem)
  - Scheduling y Timetabling (por ejemplo asignación de alumnos a PLs)
  - El problema de las N reinas
  - …
- Optimización numérica
  - Funciones max-min multimodales
  - Ajuste de parámetros
  - Generación y reconocimiento de imágenes
  - Entrenamiento de redes neuronales
  - Cálculo de estructuras
  - …

# Aplicaciones de las metaheurísticas

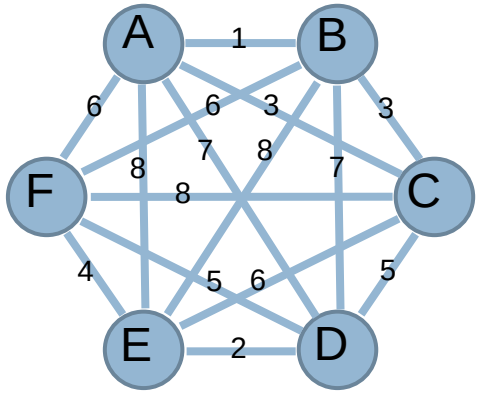
- Análisis de riesgos en inversiones o predicción de ingresos
- Calibración de modelos predictivos meteorológicos
- Aprendizaje y minería de datos
- Alineamiento de secuencias o plegado de proteínas
- Identificación de características en imágenes
- Modelado e identificación de sistemas
- Control de procesos
- Testing de software
- Clustering y aprendizaje
- Vida artificial
- Diseño generativo
- …

# Contenidos

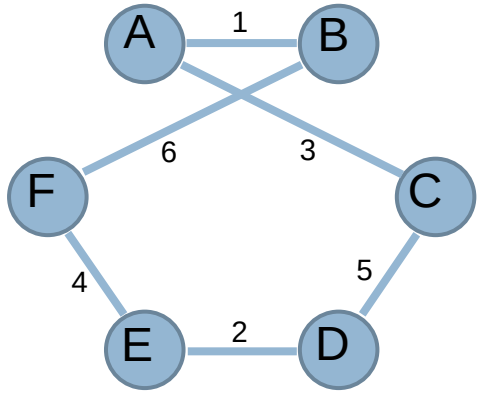
1. Metaheurísticas: introducción y clasificación
2. **Algoritmos genéticos**
   - **Algunos ejemplos motivadores**
   - Introducción y esquema general
   - El Algoritmo Genético Simple (SGA)
   - Operadores de cruce, mutación, selección y reemplazamiento
   - Los conceptos de Diversidad y Convergencia Prematura
3. Introducción a la búsqueda local
4. Algoritmos meméticos
5. Introducción a la optimización multiobjetivo
6. Conclusiones

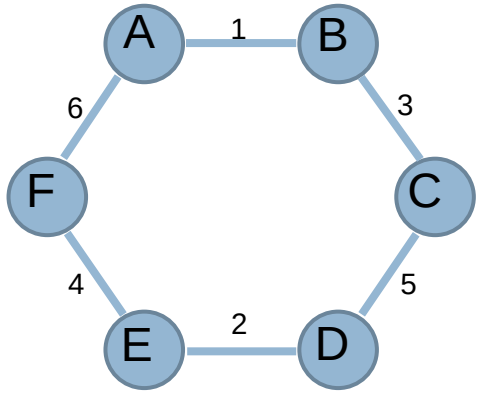
# Un ejemplo
## El problema del viajante de comercio (TSP)

- Un viajante de comercio debe visitar cada una de las *N* ciudades exactamente una vez, comenzando en una de ellas y volviendo al punto de partida, con el mínimo coste posible




Una instancia del TSP



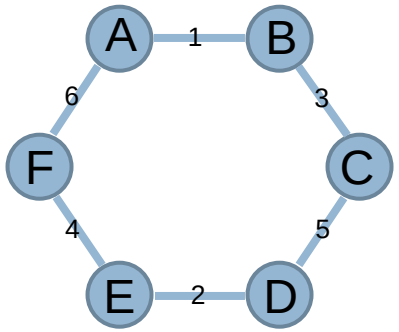




Dos soluciones óptimas (Coste = 21)

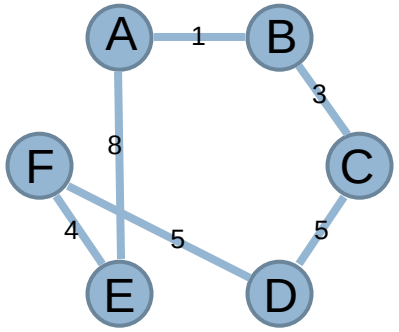
# El problema del viajante de comercio
## Cómo encontrar una solución óptima o casi óptima

- Enumerando (*N*-1)! soluciones
  - $N = 6$ $5! = 120$
  - $N = 61$ $60! > 10^{80} \cong$ número de partículas elementales en el universo
- Búsqueda inteligente exacta (algoritmo A*, ramificación y poda, …)
  - Tienen problemas con instancias muy grandes
- Otra opción: *combinar soluciones para obtener soluciones nuevas, en un proceso evolutivo de soluciones potenciales*
  - ¡Estas son las claves de los Algoritmos Genéticos!
  - La combinación de soluciones requiere:
    - Codificar las soluciones
    - Un algoritmo para combinar las codificaciones
  - Evolución (selección preferente de los mejores, variedad en la población, …)

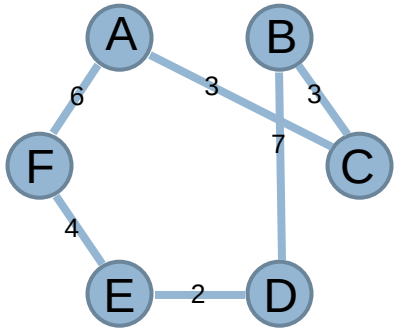
# El problema del viajante de comercio
## Codificación con permutaciones



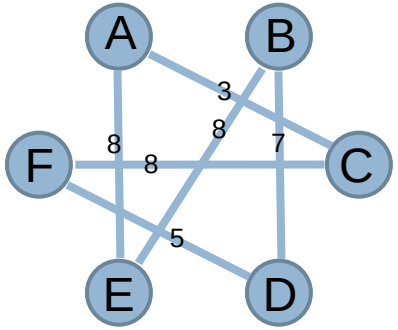

A B C D E F



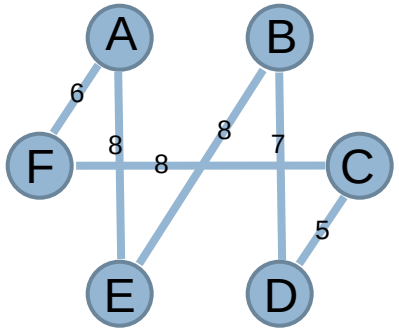

A B C D F E



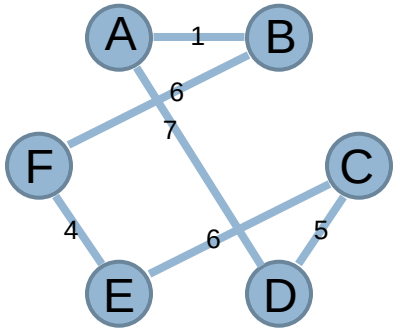

A C B D E F




A C F D B E

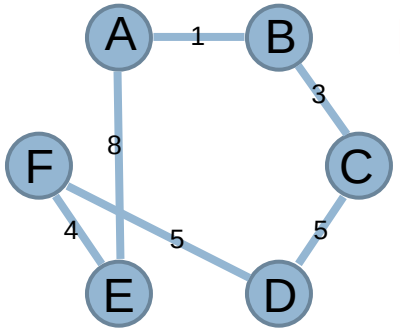



A E B D C F



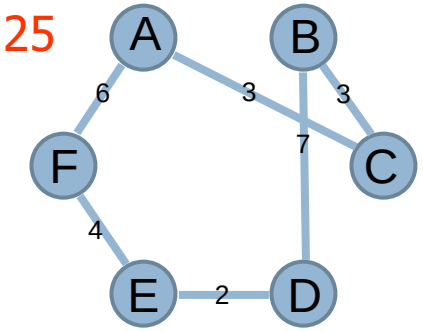

A B F E C D

# Combinando soluciones en el TSP
## Ejemplo de combinación no productiva

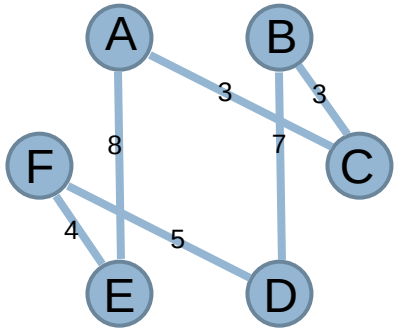



Padre_1: 26

Padre_2: 25

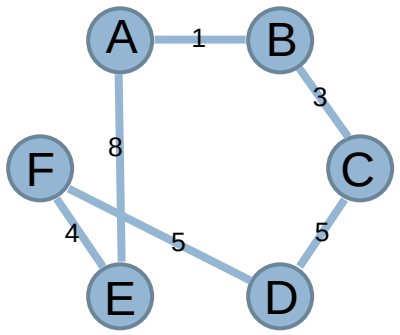



A B C D F E → A C B D F E ← A C B D E F



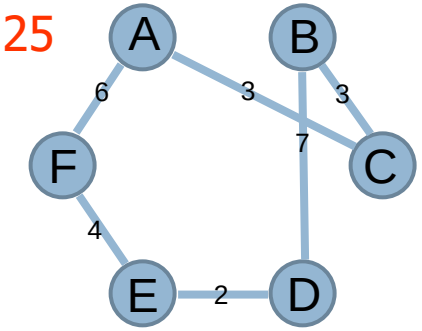

Hijo: 29

# Combinando soluciones en el TSP
## Ejemplo de combinación productiva

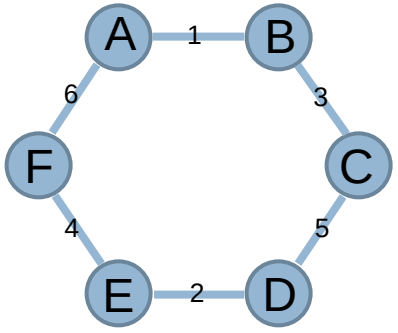



Padre_1: 26

Padre_2: 25

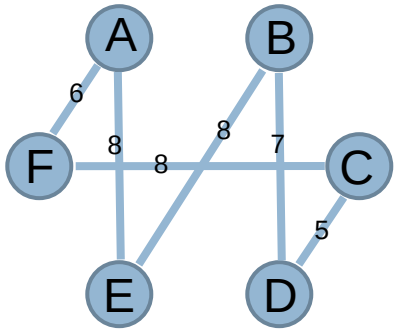



A B C D F E → A B C D E F ← A C B D E F

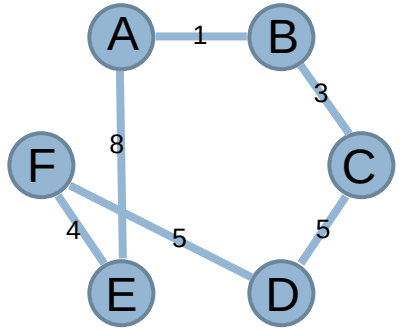



Hijo: 21

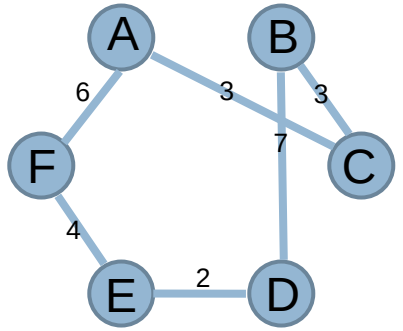
# Modificando soluciones en el TSP
## Soluciones mutadas



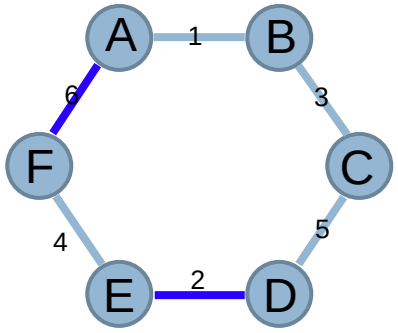

B D C F A E




B C D F E A




B C A F E D

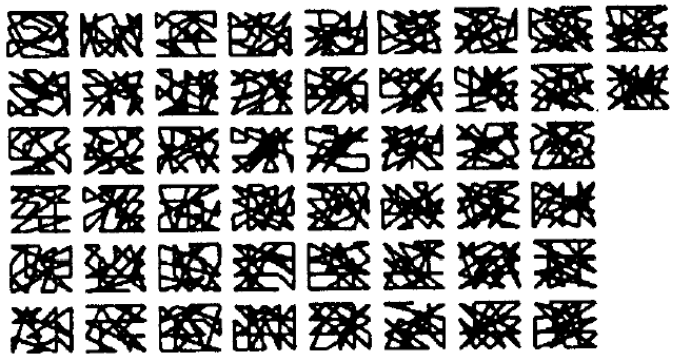



B C D E F A

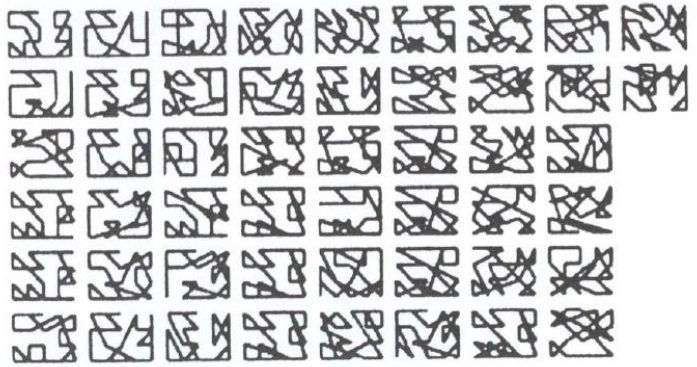
Mutada: 21

- En ocasiones, por mucho que combinemos soluciones, no lograremos la solución óptima, la inferior. Sin embargo, a partir de una solución no-optima a veces podemos modificarla levemente para alcanzar la óptima (Mutación)

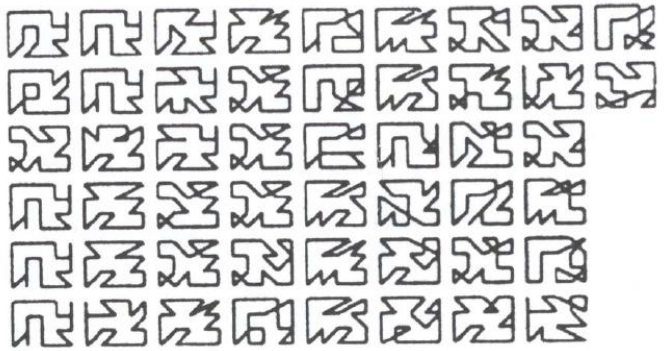
# Evolución de una población de soluciones



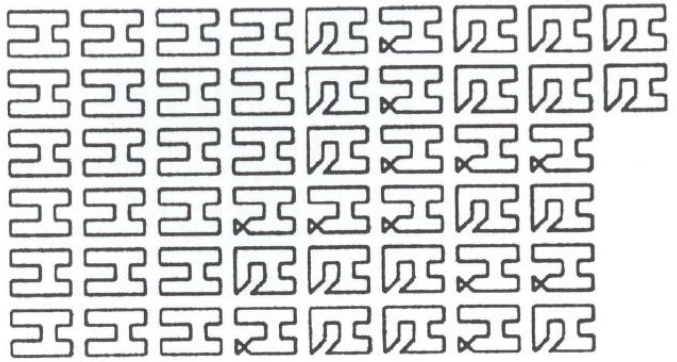
Generación 0



Generación 10

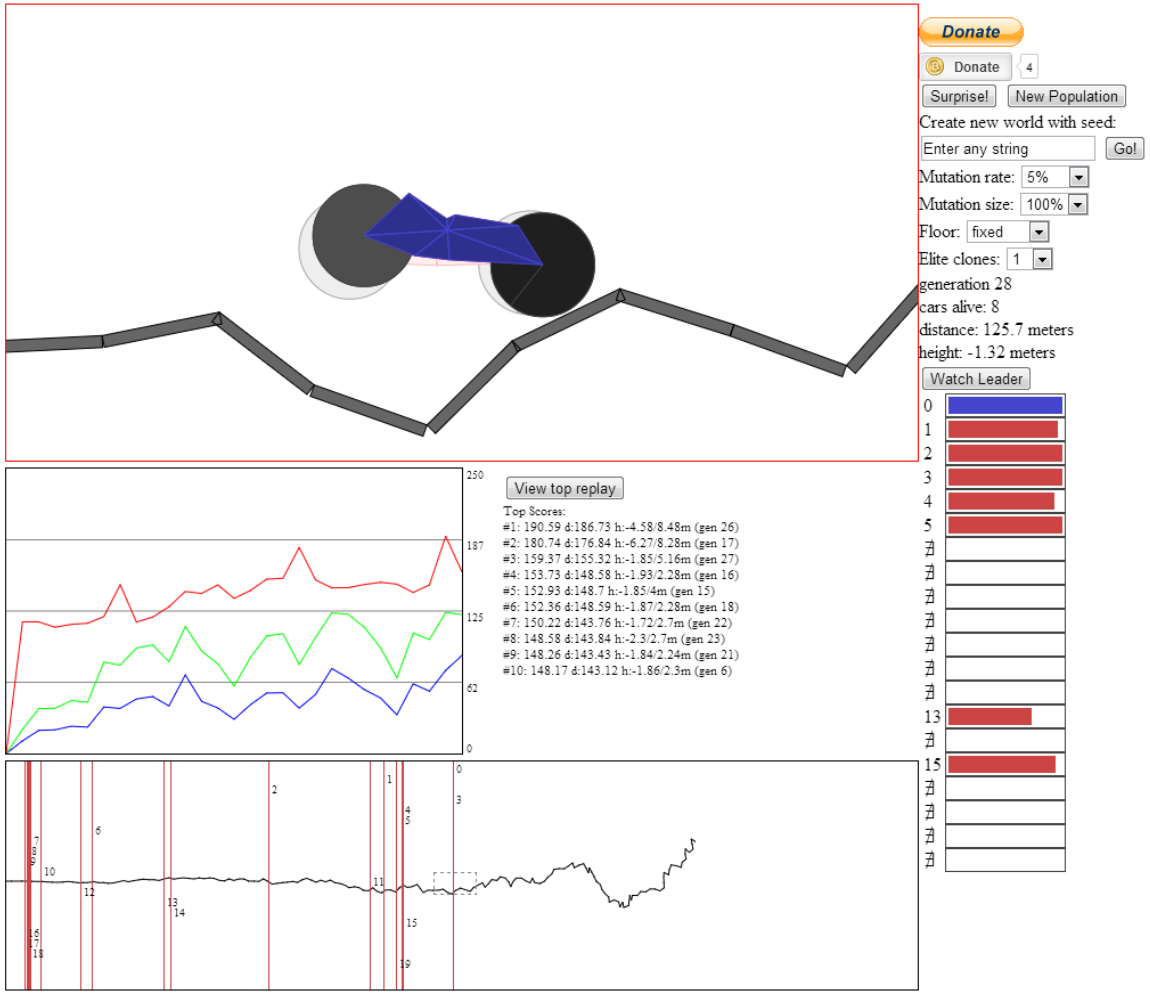


Generación 30



Generación 70

# Ejemplo
## Diseño de vehículos de dos ruedas




**2D Car HTML5**

**2D Car Flash**

**2D Car Unity**

**3D Car Unity**

**Genetic Walkers**

# Contenidos

1. Metaheurísticas: introducción y clasificación
2. **Algoritmos genéticos**
   - Algunos ejemplos motivadores
   - **Introducción y esquema general**
   - El Algoritmo Genético Simple (SGA)
   - Operadores de cruce, mutación, selección y reemplazamiento
   - Los conceptos de Diversidad y Convergencia Prematura
3. Introducción a la búsqueda local
4. Algoritmos meméticos
5. Introducción a la optimización multiobjetivo
6. Conclusiones

# Introducción

- Los Algoritmos Genéticos son algoritmos de búsqueda y optimización que están basados en mecanismos de evolución natural, en particular la selección natural y la herencia genética

- Los presentó John Holland y sus colaboradores en la Universidad de Michigan al principio de los 1970's [*Holland 1975. Adaptation in Natural and Artificial Systems*]

# Las principales características de un AG

- AGs codifican soluciones por medio de cadenas de símbolos que se denominan cromosomas
- Evolucionan una población de cromosomas, generalmente empezando de soluciones aleatorias
- La única información del dominio del problema que requiere es la del fitness o función de evaluación
- Utilizan reglas de transformación probabilistas en lugar de deterministas

# ¿Por qué han tenido éxito los AGs?

- Han resuelto problemas muy difíciles en muchas áreas de la ciencia, ingeniería, industria y administración.
- Son fáciles de comprender, diseñar e implementar
- Son flexibles, robustos y razonablemente eficientes
- No imponen restricciones a la función objetivo (continuidad, derivabilidad, multimodalidad, …)
- Son bastante divertidos
- . . .

# La metáfora evolutiva

| Evolución Natural | Evolución Artificial |
|---|---|
| ▪ Ecosistema y Entorno<br>▪ Cromosoma y Genotipo<br>▪ Individuo y Fenotipo<br>▪ Adaptación al Entorno<br>▪ Supervivencia, Reproducción, Mutación | ▪ Instancia del Problema<br>▪ Cadena de Símbolos<br>▪ Solución del Problema<br>▪ Fitness o Función Objetivo<br>▪ Selección, Cruce, Mutación, Reemplazamiento |

# Los principales componentes de un AG

- Un esquema de codificación, por ejemplo cadenas de dígitos binarios (0 y 1)
- Una función de fitness que valore la calidad de los cromosomas
- Algún método para generar la población inicial: *aleatorio, heurístico, …*
- Un conjunto de operadores genéticos: *selección, cruce, mutación, reemplazamiento, …*
- Un conjunto de parámetros: *probabilidad de cruce, probabilidad de mutación, tamaño de la población, número de generaciones, …*

# Un algoritmo genético convencional
## Estructura

```
Algoritmo Genético
    Parámetros de entrada (ProbCruce, ProbMutacion, maxGen, PobSize, … );
    numGen ← 0;
    Inicializar(Pob(0));                                  // Población inicial
    Evaluar(Pob(0));                                      // Función de fitness
    while ( numGen < maxGen )  {                          // Condición de parada
        numGen ← numGen+1;
        Pob'(numGen) = Selección(Pob(numGen-1));                          // Selección
        Pob''(numGen) = Cruce(Pob'(numGen));                              // Cruce
        Pob'''(numGen) = Mutación(Pob''(numGen));                         // Mutación
        Evaluar(Pob'''(numGen));                                          // Función de fitness
        Pob(numGen) = Reemplazo(Pob'(numGen), Pob'''(numGen));            // Reemplazo
    }
    return el mejor individuo en Pob(maxGen);
end
```

# El esquema de codificación

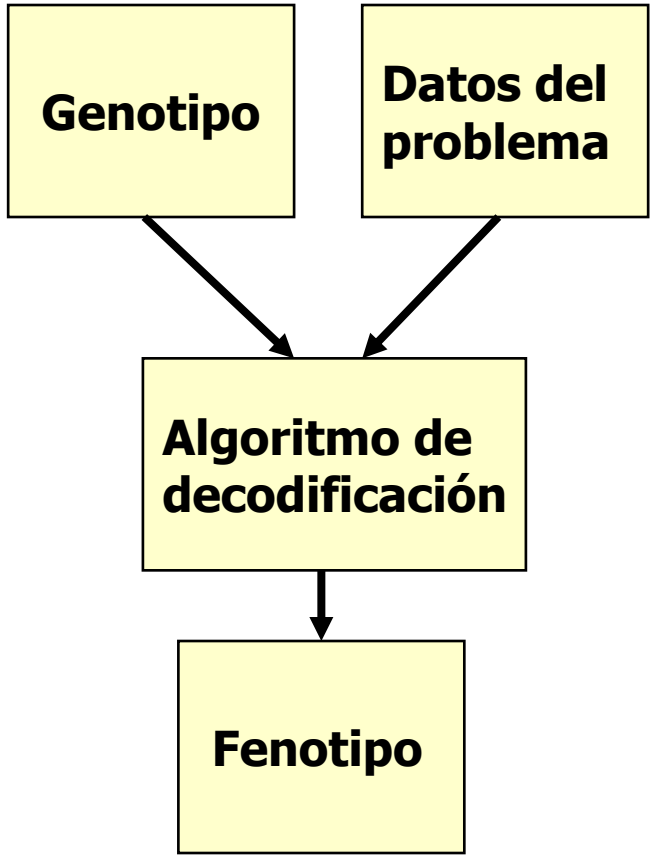
- Codifica una potencial solución al problema, y el algoritmo de evaluación la debe construir de forma eficiente (en tiempo polinomial)
- Debe tener buena capacidad de representación
  - Al menos de un subconjunto de soluciones buenas
  - A ser posible con una correspondencia de uno a uno
- Debe ser fácil diseñar operadores genéticos tales que
  - Generen cromosomas factibles
  - Trasladen características relevantes de padres a hijos
  - No sean demasiado costosos computacionalmente
  - Los hijos tengan una "alta" probabilidad de mejorar

# El esquema de codificación
## Algunas codificaciones típicas

- Cadenas de dígitos binarios
- Vectores de números reales (Optimización numérica)
  - Optimización de funciones ($f:R^n \rightarrow R$)
  - Optimización de parámetros de sistemas físicos
- Permutaciones de símbolos (Optimización combinatoria)
  - Problema del viajante de comercio: permutación de las ciudades (5 3 4 1 2)
  - Scheduling: permutación de las tareas (1 3 2 4 5)
- Otras estructuras más complejas
  - Matrices
  - Árboles (Programación Genética)
  - . . .

# El algoritmo de decodificación

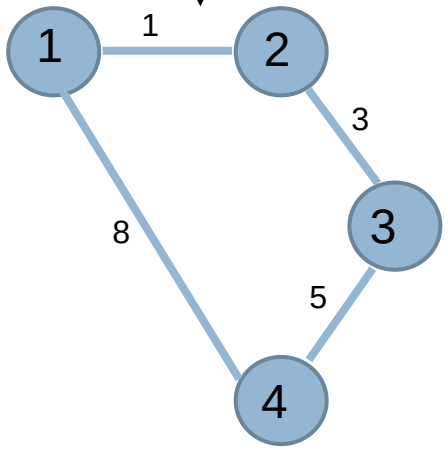
- Construye el fenotipo a partir del genotipo
- Suele ser el componente más costoso computacionalmente
- A menudo se utiliza un algoritmo voraz (por ejemplo en optimización combinatoria)
- El fitness se obtiene del fenotipo



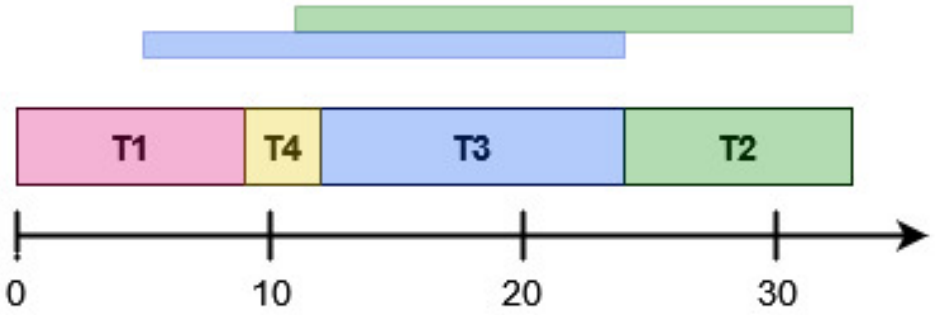
# El algoritmo de decodificación
## Ejemplo utilizando permutaciones

(1 4 3 2)

Decodificador Traveling Salesman Problem




Decodificador One Machine Scheduling Problem



# Contenidos

1. Metaheurísticas: introducción y clasificación
2. **Algoritmos genéticos**
   - Algunos ejemplos motivadores
   - Introducción y esquema general
   - **El Algoritmo Genético Simple (SGA)**
   - Operadores de cruce, mutación, selección y reemplazamiento
   - Los conceptos de Diversidad y Convergencia Prematura
3. Introducción a la búsqueda local
4. Algoritmos meméticos
5. Introducción a la optimización multiobjetivo
6. Conclusiones

# El Algoritmo Genético Simple (SGA)

- Codificación binaria (cadenas de bits)
- Inicialización aleatoria de la población
- Cruce en un punto
- Mutación simple
- Selección proporcional al Fitness (regla de la ruleta)
- Reemplazo generacional con aceptación incondicional (los hijos reemplazan a sus padres)

# Codificación binaria

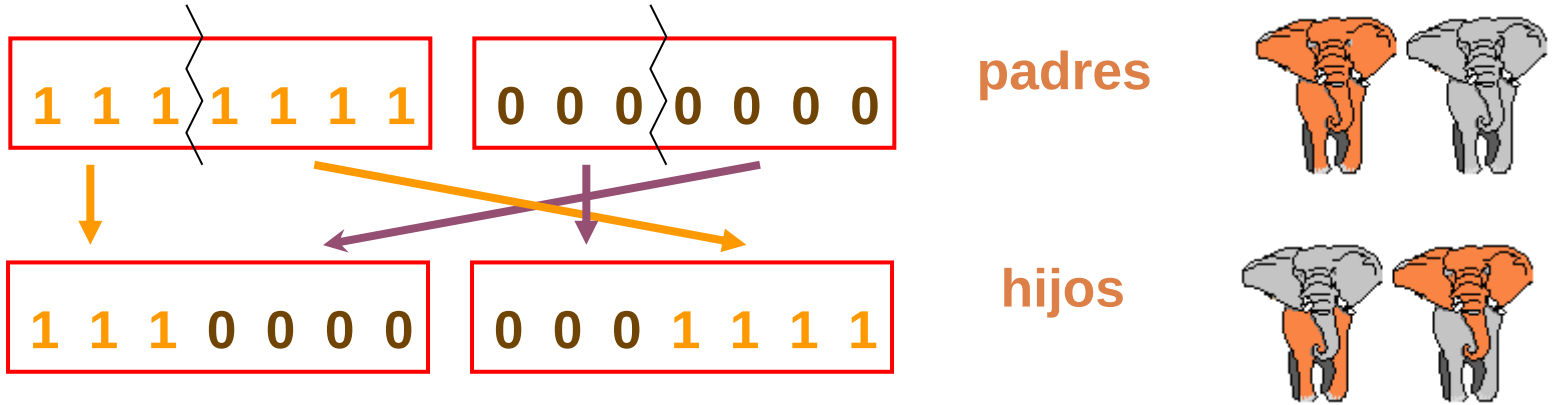
- Un cromosoma es una cadena binaria, cada bit es un gen

| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|

- La solución que codifica depende tanto del problema como del algoritmo de decodificación
  - Problema 1: Maximizar $f : [a,b] \subset R \rightarrow R^+$
    - Una posible solución es un número real $x \in [a,b]$
  - Problema 2: Repartir 8 trabajos entre 2 máquinas
    - S = {T1 M1, T2 M0, T3 M1, T4 M0, T5 M0, T6 M0, T7 M1, T8 M1}
- El valor del fitness es la calidad de la solución
  - Problema 1: el valor f(x)
  - Problema 2: el beneficio obtenido aplicando la planificación S

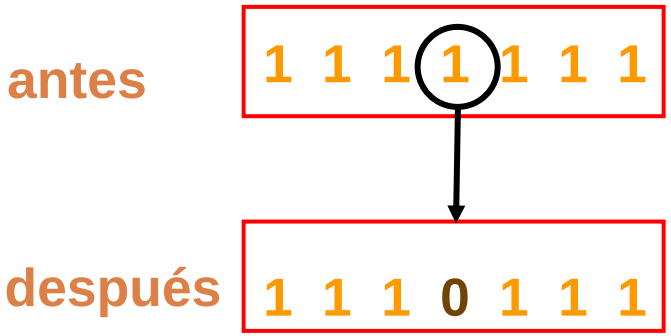
# Cruce en un punto

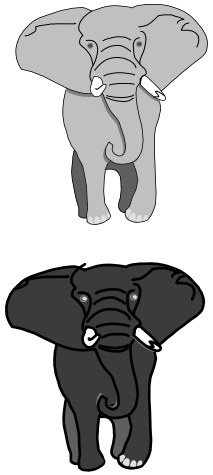
- Genera dos hijos de dos padres (se suele aplicar con una alta probabilidad Pc: 0.6 a 1)
- Cada hijo hereda características de sus padres
- Es el componente de EXPLOTACIÓN y es posiblemente el operador más importante de un Algoritmo Genético



# Mutación simple

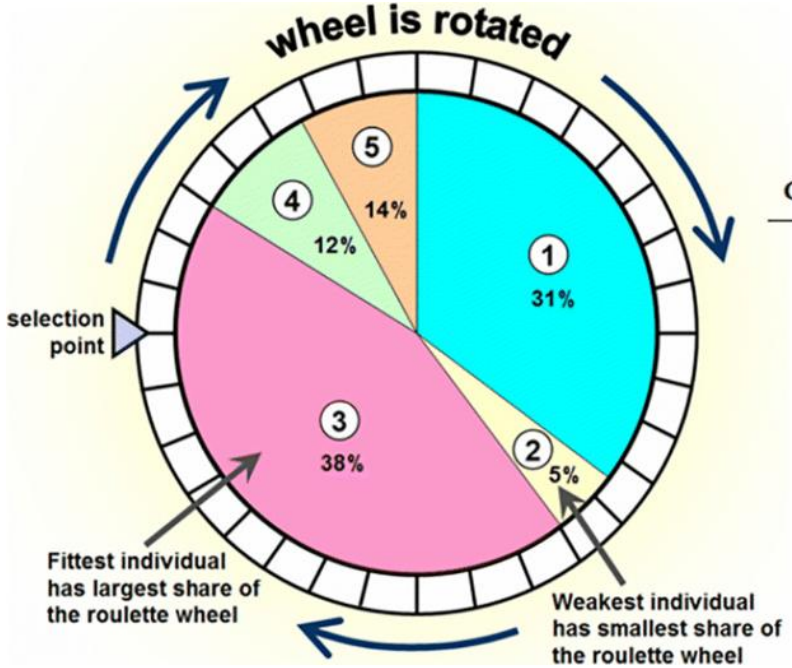
- Modifica un único bit (gen) según la probabilidad de mutación (muy pequeña, alrededor de 0.01)
- Introduce características aleatorias en la estructura del cromosoma. De esta forma es posible introducir nuevo material genético en la población
- Es el componente de EXPLORACIÓN



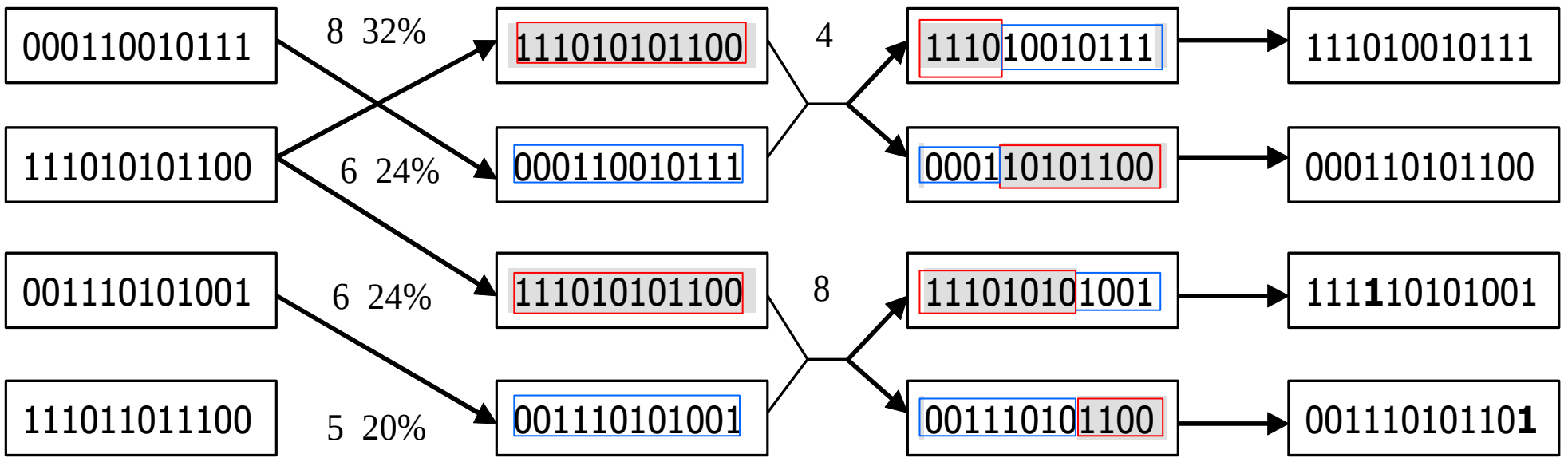
# Selección por ruleta

- La probabilidad de selección del cromosoma *i* es $f_i / \sum_{i=1..n} f_i$
- Se aplica *PopSize* veces para generar *Pob'()* a partir de *Pob()*
- Es necesario ajustar la presión selectiva
  - Presión alta: convergencia prematura
  - Presión baja: no converge
  - Una solución: Escalar el fitness
  - Un inconveniente: Depende del problema




| Chromosome | Fitness value |
|---|---|
| C1 | 31% |
| C2 | 5% |
| C3 | 38% |
| C4 | 12% |
| C5 | 14% |

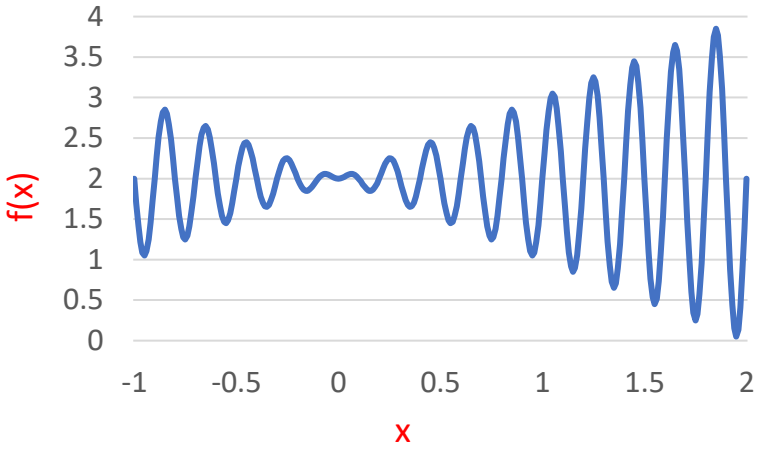
# Un ejemplo de selección, cruce y mutación



# Ejemplo de aplicación
## Optimización numérica

- Ejemplo: calcular el máximo de una función en un intervalo

$$f(x) = x*\sin(10\pi x)+2.0 \text{ en } [-1,2]$$




- Solución con un AG:
  - Posibles soluciones: $x \in [-1,2]$
  - Decodificación: $s \rightarrow x$; $(0\ 0\ \ldots\ 0) \rightarrow -1$; $(1\ 1\ \ldots\ 1) \rightarrow 2$
  - Fitness(s) = f(Decodificación(s))
  - La longitud del cromosoma depende de la precisión deseada, y determina el número de posibles cromosomas diferentes
    - Si, por ejemplo, queremos una precisión de $10^{-9}$, necesitaríamos poder generar 3000 millones de cromosomas diferentes (1000 millones por cada unidad, multiplicado por 3 unidades de rango), por lo que su longitud debe ser 32, ya que $2^{31} < 3000000000 < 2^{32}$

# Ejemplo de aplicación
## Optimización combinatoria

- Ejemplo: problema de la mochila
  - Datos de entrada: valor y peso de cada objeto, y la capacidad máxima de la mochila
  - Objetivo: seleccionar un conjunto de objetos que maximicen el valor
  - Restricción: debemos respetar la capacidad máxima de la mochila
- Solución con un AG:
  - El cromosoma será una cadena de 0's y 1's de longitud igual al número de objetos
  - Decodificación: lo más intuitivo sería que con un 1 se mete ese objeto en la mochila, y con un 0 no se mete
  - Problema: ¿cómo hacer que se cumpla la restricción?
    - Una posible solución: recorrer el cromosoma, y si la posición es 1, meter ese objeto solo si metiéndolo no se supera la capacidad máxima de la mochila

# Contenidos

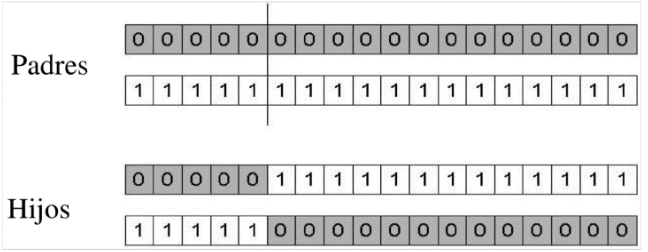
1. Metaheurísticas: introducción y clasificación
2. **Algoritmos genéticos**
   - Algunos ejemplos motivadores
   - Introducción y esquema general
   - El Algoritmo Genético Simple (SGA)
   - **Operadores de cruce, mutación, selección y reemplazamiento**
   - Los conceptos de Diversidad y Convergencia Prematura
3. Introducción a la búsqueda local
4. Algoritmos meméticos
5. Introducción a la optimización multiobjetivo
6. Conclusiones

# Operadores de cruce

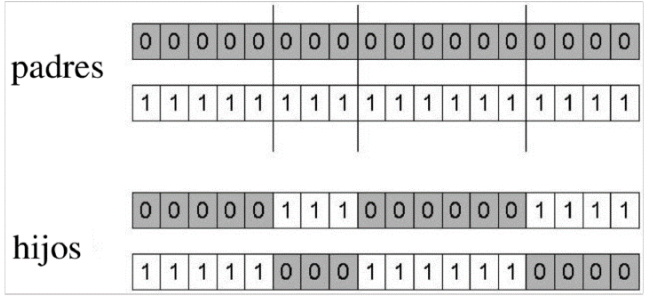
- Algunos aspectos importantes:
  - Los hijos deben heredar características relevantes de los padres
  - Se debe diseñar de acuerdo a la representación elegida para el problema
  - Debe producir cromosomas válidos
  - No debería ser muy costoso computacionalmente
  - Probabilidad de cruce típica: entre 60% y 100% (en general alta)
  - Lo habitual es cruzar dos padres y generar dos hijos, pero también existen operadores de cruce que cruzan más de dos padres, o que generan un único hijo, o que generan un mayor número de hijos
- Algunos ejemplos de operadores de cruce:
  - Para representación binaria:
    - Cruce en 1 punto, cruce en *n* puntos, cruce uniforme, …
  - Para representación mediante números reales
    - Cruce aritmético simple, cruce BLX-α, …
  - Para representación mediante permutaciones
    - Order crossover (OX), Partially Mapped Crossover (PMX), Cycle Crossover (CX), …

# Operadores de cruce
## Para representación binaria

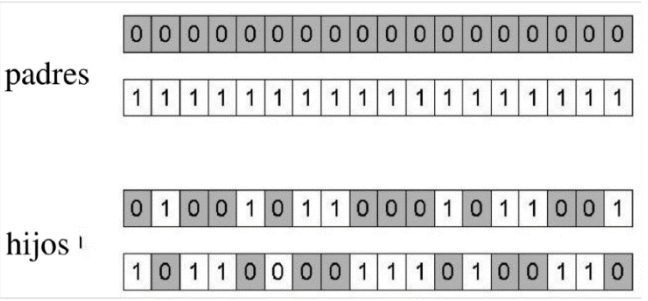
- Cruce en 1 punto
  - Seleccionar un punto al azar en ambos padres
  - Cortar los padres en este punto
  - Crear hijos intercambiando las “colas”




- Cruce en *n* puntos
  - Seleccionar n puntos al azar en ambos padres
  - Cortar los padres en estos puntos
  - Pegar las partes alternando entre los padres
  - Generalización del cruce en 1 punto




- Cruce uniforme
  - Número aleatorio [0, 1] para cada gen
  - En base a dicho aleatorio, tomar el gen de uno u otro padre
  - El otro hijo será el inverso



# Operadores de cruce
## Para representación real

- Cruce aritmético simple
  - Cada gen de la descendencia toma el valor medio de los genes de los padres
  - Tiene la desventaja de que únicamente se genera un descendiente
  - Otra desventaja es que iremos perdiendo diversidad poco a poco, al ir centrándonos en los puntos medios

| 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 |
|---|---|---|---|---|---|

# Operadores de cruce
## Para representación real

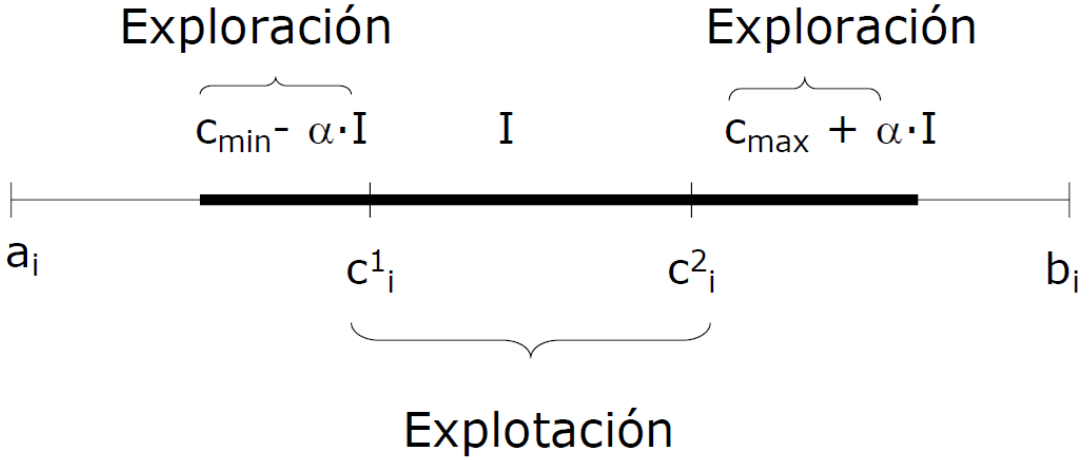
- Cruce BLX-α
  - Dados 2 cromosomas: $C_1 = (c_{11}, \dots, c_{1n})$ y $C_2 = (c_{21}, \dots, c_{2n})$
  - BLX-α genera descendientes de la forma $(h_1, \dots, h_i, \dots, h_n)$, tales que $h_i$ se genera aleatoriamente en el intervalo: $[C_{min} - I \cdot \alpha,\ C_{max} + I \cdot \alpha]$

    en donde

    $C_{max} = \max \{c_{1i}, c_{2i}\}$

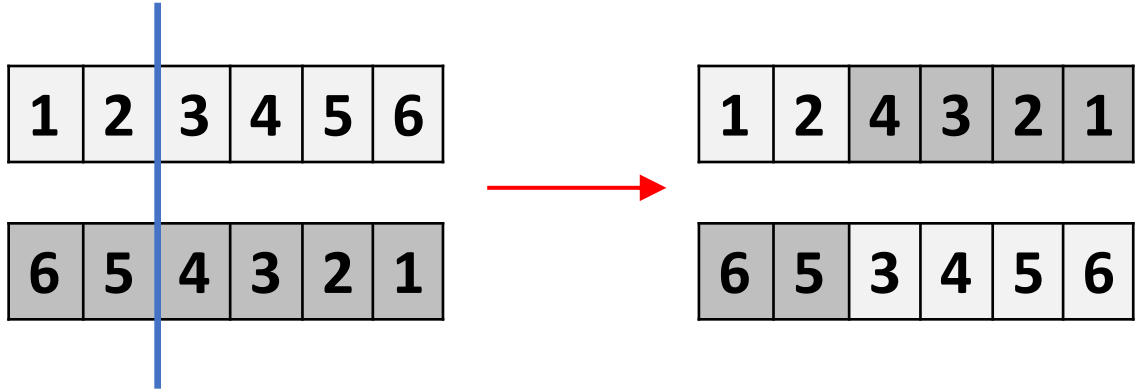
    $C_{min} = \min \{c_{1i}, c_{2i}\}$

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



# Operadores de cruce
## Para permutaciones

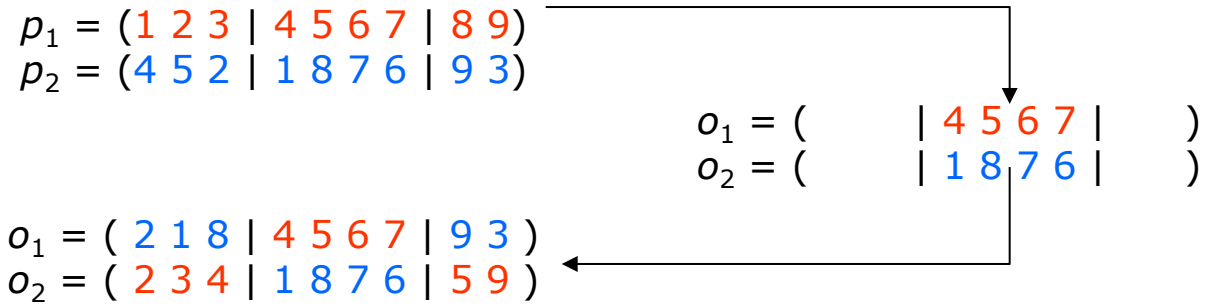
- Aplicar un operador de cruce estándar puede llevar a soluciones no factibles. Por ejemplo:




- Algunos posibles operadores de cruce son:
  - Order Crossover (OX)
  - Partially Mapped Crossover (PMX)
  - Cycle Crossover (CX)
  - …

# Operadores de cruce
## Para permutaciones

- Order Crossover (OX): Cruce basado en orden
  - Una subsecuencia, elegida aleatoriamente, se copia de uno de los padres al hijo, y el resto de los genes se copian en el resto de posiciones preservando el orden relativo en el que estén en el otro padre
  - Un segundo hijo se puede crear invirtiendo el rol de los padres
  - Ejemplo:




  - Cada descendiente hereda el orden y posición de algunos genes de uno de los padres y el orden relativo del resto de genes del otro padre
  - Existe una modificación de este operador de cruce en el que, en lugar de elegir una subsecuencia, se eligen genes al azar

# Operadores de mutación

- Algunos aspectos importantes:
  - Debería permitir alcanzar cualquier parte del espacio de búsqueda
  - El tamaño de la mutación debe ser controlado
  - Se debe diseñar de acuerdo a la representación elegida para el problema
  - Debe producir cromosomas válidos
  - No debería ser muy costoso computacionalmente
  - Probabilidad de mutación típica: entre 0% y 30% (en general muy baja)
- Algunos ejemplos de operadores de mutación:
  - Mutación clásica (seleccionar uno o varios genes y cambiar sus valores)
  - Mutación uniforme (para representación real o entera)
  - Mutación basada en intercambios, reordenación, inserción, inversión
  - Mutación BGA (para codificación real)

# Operadores de mutación

- Mutación clásica / estándar
  - Se selecciona uno o varios genes y se cambian sus valores
    - Un único gen

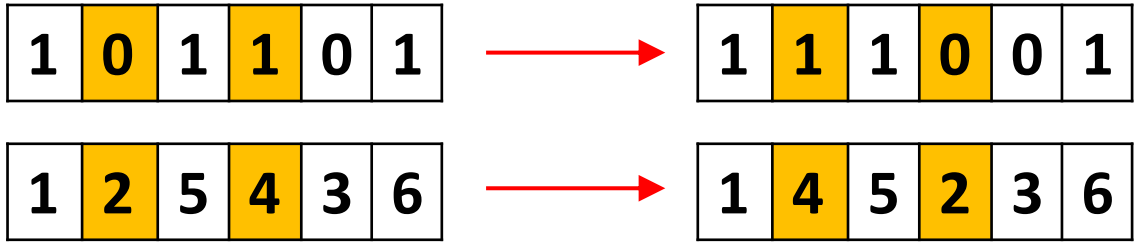

    - Múltiples genes


    - Cada gen con una probabilidad $p_m$

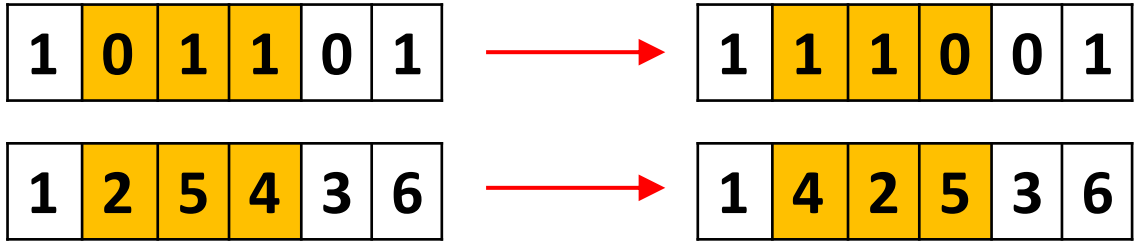

- Mutación uniforme (para representación real/entera)
  - Se selecciona un gen y su valor (entero o real) es escogido aleatoriamente entre los valores de un rango


# Operadores de mutación

- Mutación basada en intercambios
  - Se seleccionan dos genes y se intercambian sus valores




- Mutación basada en reordenación
  - Se selecciona un subconjunto de genes y sus valores son reordenados de manera aleatoria



# Operadores de mutación

- Mutación basada en inserción
  - Selecciona aleatoriamente un gen y lo inserta en otro lugar al azar


  - Puede realizarse para un conjunto de genes

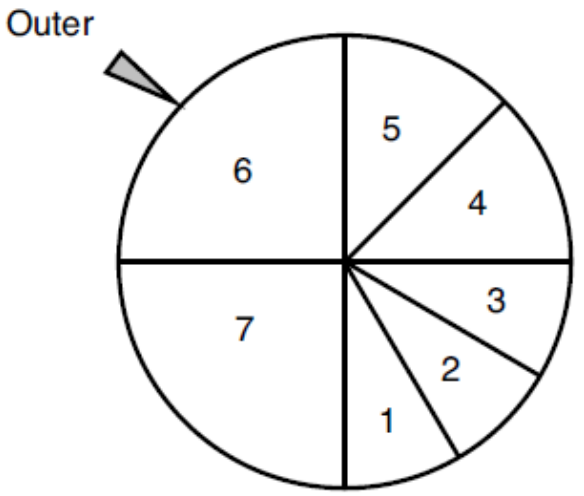

- Mutación basada en inversión
  - Selecciona un conjunto de genes y los inserta de manera inversa


  - Puede realizarse insertándola en otra posición

# Operadores de selección

- Selección por ruleta
  - Las soluciones buenas tienen mayor probabilidad de reproducirse que las malas, siendo la probabilidad proporcional al fitness
  - Todos los cromosomas tienen alguna probabilidad de reproducirse
  - Inconveniente: Requiere escalar el fitness para ajustar la presión selectiva. El escalado es dependiente del problema

| Individuals: | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Fitness: | 1 | 1 | 1 | 1.5 | 1.5 | 3 | 3 |




Roulette selection

# Operadores de selección

## ¿Por qué se debe escalar el fitness en ruleta?

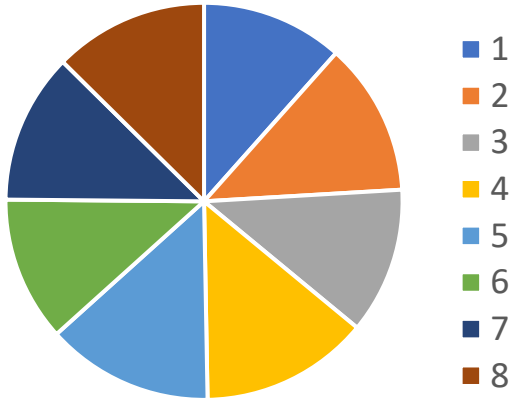
- Ejemplo: consideremos que, en un problema de maximización, tenemos 8 individuos en la población con los siguientes valores de fitness:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 1020 | 1100 | 1050 | 1210 | 1200 | 1040 | 1080 | 1110 |

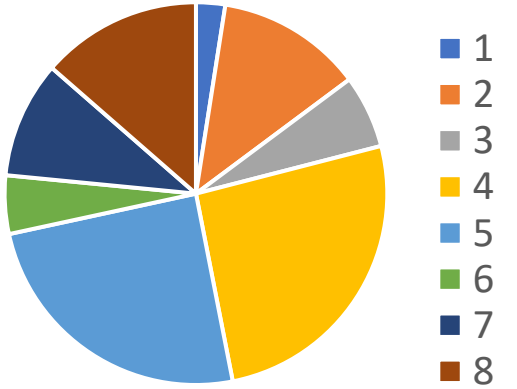
- ¡Estaríamos dando casi la misma probabilidad a todos!
- Pero… ¿qué ocurriría si hacemos un escalado, por ejemplo, restando 1000 a cada fitness?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 20 | 100 | 50 | 210 | 200 | 40 | 80 | 110 |

- Ahora ya podemos discriminar mejor entre los mejores individuos y los peores
- Problema: hay muchas formas de escalar… ¿cuál es mejor?




Sin escalado




Con escalado

# Operadores de selección
## Operadores en los que no hay que escalar el fitness

- Emparejamiento aleatorio (mating): todos los individuos de la población son padres una vez, y solo una vez. Es decir, cada individuo se selecciona una vez. En este caso, se suele aplicar la presión selectiva en la fase de reemplazamiento, haciendo que sobrevivan los dos mejores de entre cada pareja de padres y sus dos respectivos hijos.
- Torneo m:n
  - En general, un torneo m:n consiste en tomar *m* cromosomas al azar de la población y de esos *m* se seleccionan los *n* mejores
  - En la práctica, lo más habitual es utilizar un torneo m:1 para elegir a cada padre
  - Elegir bien el tamaño del torneo (m) es importante: valores muy bajos harán una selección casi aleatoria, mientras que valores elevados harán demasiada presión selectiva

# Operadores de reemplazamiento

- Incondicional (o generacional)
  - Los progenitores son reemplazados por sus descendientes
  - Utilizado habitualmente cuando se aplica selección por ruleta o torneo
- Los mejores de entre cada pareja/grupo de progenitores y sus correspondientes descendientes
  - Por ejemplo, elegir los 2 mejores de entre cada grupo de 2 progenitores y sus respectivos 2 descendientes
  - Variante: elegir los 2 mejores pero que sean diferentes (mejora la diversidad)
  - Utilizado habitualmente cuando se aplica emparejamiento aleatorio

# Otros operadores

- Elitismo
  - Los mejores individuos en P(t) pasan sin cambios a P(t+1)
  - Suele ser algo muy aconsejable
  - Ya está implícito en el segundo operador de reemplazamiento descrito
- Población sin duplicados
  - Ventaja: mejora la diversidad
  - Inconveniente: el reemplazamiento conlleva un importante consumo de tiempo
- Búsqueda local
  - Además de evaluar cada individuo, le aplicaremos algún algoritmo de búsqueda local, para mejorarlo
  - En este caso tendríamos un algoritmo híbrido, denominado algoritmo memético
  - Mejora la intensificación

# Condición de parada

- Tenemos varias opciones para finalizar la ejecución de un algoritmo genético:
  - Alcanzar el óptimo
    - Inconveniente: en muchas ocasiones no sabremos cuál es
  - Tiempo máximo de ejecución
  - Número máximo de generaciones
  - Número máximo de evaluaciones
  - Número máximo de generaciones consecutivas sin mejorar la mejor solución
    - Ventaja: criterio flexible, ya que en instancias fáciles el algoritmo tardará poco tiempo, y en instancias difíciles le dejaremos evolucionar mucho más tiempo

# Contenidos

1. Metaheurísticas: introducción y clasificación
2. **Algoritmos genéticos**
   - Algunos ejemplos motivadores
   - Introducción y esquema general
   - El Algoritmo Genético Simple (SGA)
   - Operadores de cruce, mutación, selección y reemplazamiento
   - **Los conceptos de Diversidad y Convergencia Prematura**
3. Introducción a la búsqueda local
4. Algoritmos meméticos
5. Introducción a la optimización multiobjetivo
6. Conclusiones

# Convergencia de un AG

- Un AG se espera que tenga una convergencia hacia soluciones cada vez mejores a lo largo de las generaciones
- Las claves son la diversidad, la presión selectiva y la disrupción

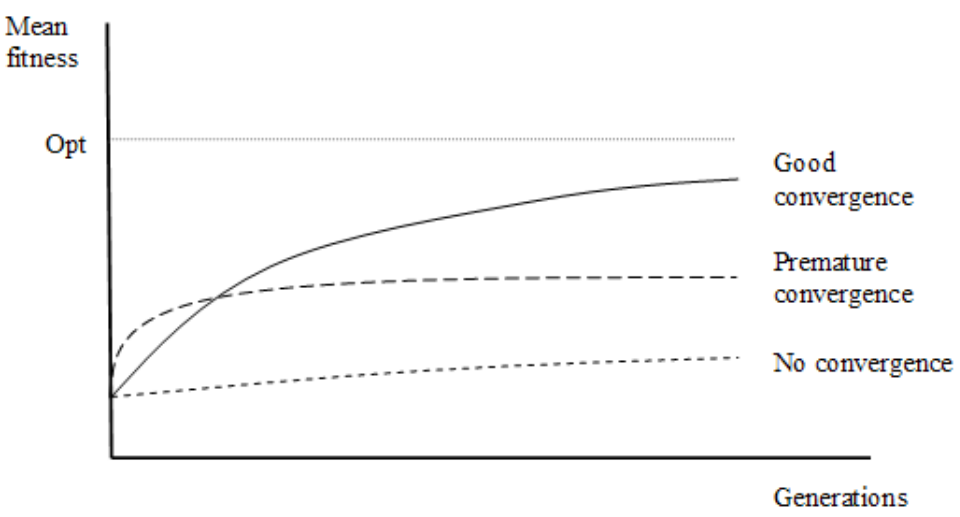
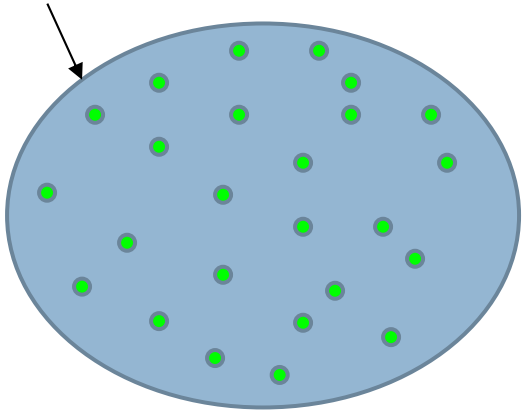



**Figure 3**. Three typical convergence patterns of the mean fitness of the population in a genetic algorithm.

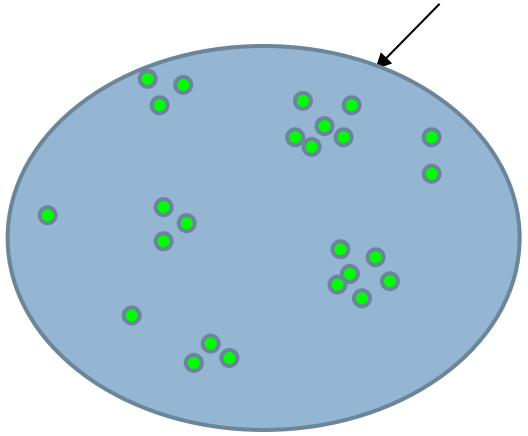
# Una de las claves
## Equilibrio entre diversificación e intensificación

- Diversificación (o exploración): muestrear regiones desconocidas.
  - Pero si hay exceso: búsqueda aleatoria, sin convergencia. Demasiada diversidad.
- Intensificación (o explotación): muestrear más intensamente en la proximidad de las soluciones buenas.
  - Pero si hay exceso: tendencia a caer en máximos locales y convergencia prematura.

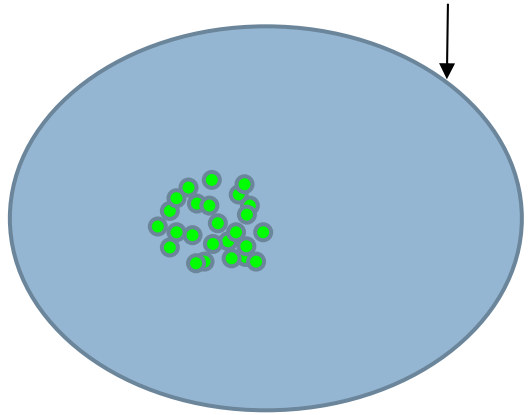
**Exceso de diversificación: Búsqueda aleatoria**



**Equilibrio: se exploran varias zonas prometedoras**



**Exceso de intensificación: caída en un máximo local**



# Claves para una buena convergencia
## Diversidad

- Diversidad del material genético: evita la convergencia prematura
  - Generación de soluciones iniciales
    - Aleatorias: alta diversidad / baja calidad
    - Heurísticas: alta calidad / baja diversidad
  - Mutación: Introduce material genético nuevo
    - Tasa de mutación demasiado baja: favorece la convergencia prematura
    - Tasa de mutación demasiado alta: búsqueda al azar (sin convergencia)
    - Opciones:
      - Tasa de mutación fija
      - Adaptar la tasa de mutación a medida que avanza la ejecución (por ejemplo, aumentando la tasa si vemos que la ejecución se estanca)
      - Aplicar probabilidades de mutación más altas a soluciones malas

# Claves para una buena convergencia
## Diversidad

- Diversidad del material genético: evita la convergencia prematura
  - Operador de cruce
    - Existen algunas técnicas de emparejamiento que permiten aumentar la diversidad en la población, por ejemplo, hacer que dos padres se crucen solo si la distancia entre ellos está por encima de cierto umbral. Cuestiones: ¿cómo medir la distancia? ¿cómo elegir el umbral?
  - Existen otros operadores para preservar la diversidad
    - Por ejemplo, reinicializar la población cuando llegamos a un punto de estancamiento.
    - Podemos detectar un punto de estancamiento cuando llevamos un cierto número de generaciones consecutivas sin mejorar la mejor solución
    - Al reinicializar la población, podemos optar por mantener un cierto número de las mejores soluciones

# Claves para una buena convergencia
## Presión selectiva y disrupción

- Una adecuada presión selectiva
  - Determina la influencia de los mejores cromosomas
  - Depende de los operadores de Selección/Remplazamiento
  - Los buenos cromosomas deben tener una mayor probabilidad de reproducirse
  - Los malos cromosomas pueden tener buen "material genético"
- No ser excesivamente disruptivos
  - Es el efecto que se produce al fragmentar buenos esquemas por el cruce y la mutación
  - Es menor en operadores específicamente diseñados para un problema frente a operadores independientes del problema

# Ejemplo: influencia de la población inicial

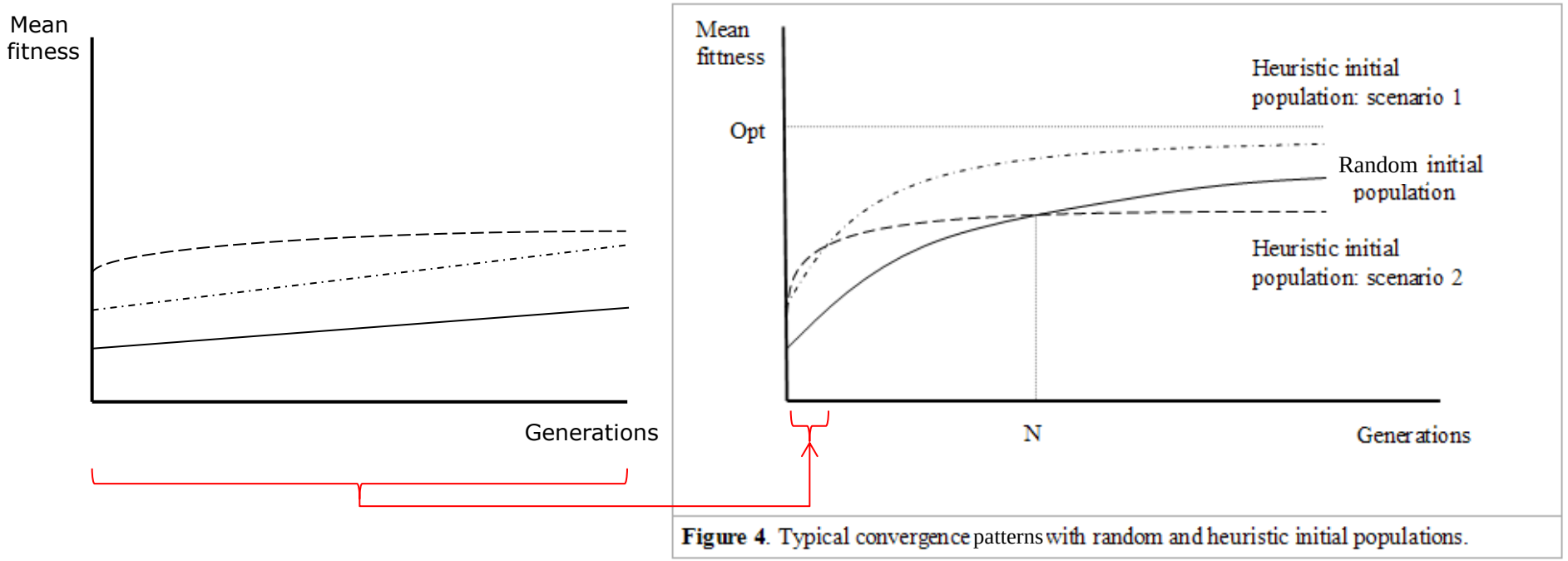



**Figure 4**. Typical convergence patterns with random and heuristic initial populations.

- Un adecuado equilibrio entre calidad y diversidad en la población inicial es necesario para una buena convergencia. En la práctica hay que experimentar para encontrar ese punto de equilibrio.
- Una opción es utilizar una mezcla de individuos heurísticos y aleatorios.

# Comparando diferentes configuraciones
## Consideraciones sobre la experimentación

- Nunca sacar conclusiones de única ejecución, debido al carácter estocástico de los algoritmos genéticos
- Si queremos comparar dos configuraciones diferentes de un algoritmo genético (por ejemplo comparar dos operadores de cruce diferentes), deberíamos:
  - Que ambas configuraciones consuman un tiempo de ejecución similar (ajustar el tamaño de población y el número de generaciones si fuese necesario)
  - Realizar un número suficiente de ejecuciones independientes (entre 5 y 30 es un buen número) con cada configuración
  - Utilizar medidas estadísticas (medias, medianas, etc.) para comparar los resultados
  - Podemos realizar algún test estadístico como el Wilcoxon para muestras pareadas

# Contenidos

1. Metaheurísticas: introducción y clasificación
2. Algoritmos genéticos
3. **Introducción a la búsqueda local**
4. Algoritmos meméticos
5. Introducción a la optimización multiobjetivo
6. Conclusiones

# Introducción

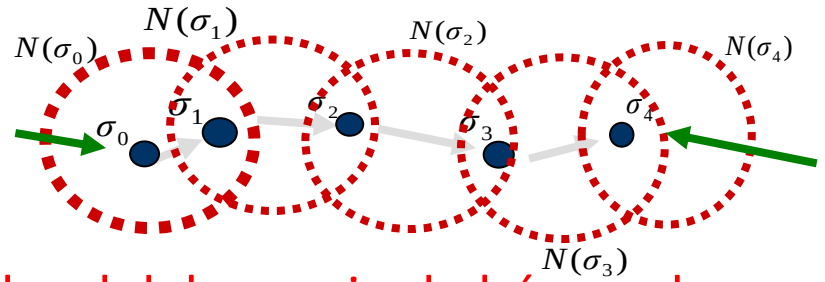
## ¿A qué nos referimos con "local" en la búsqueda local?

- El término "local" se suele utilizar en relación con las metaheurísticas de búsqueda. Se asocia a la utilización de estructuras de entorno o vecindad, refiriéndose al concepto de proximidad o vecindad entre diferentes soluciones del problema

- Todas las soluciones incluidas en el entorno de la solución actual, que viene delimitado por un operador de generación de soluciones, se denominan soluciones vecinas

- Las metaheurísticas basadas en trayectorias comienzan en una solución del problema, y analizan su entorno para decidir cómo continuar el recorrido de la búsqueda

Solución inicial




Óptimo local/global

- Efectúan un estudio local del espacio de búsqueda

# Introducción
## Estructura de entorno o vecindad

- Una solución vecina *s'* se obtiene modificando algo de la solución actual *s* utilizando algún operador de entorno o vecindad
- El operador de vecindad se suele definir utilizando el concepto de movimiento, que modifica uno o más atributos de una solución dada para generar otra solución
- El conjunto de todos los posibles cambios define la vecindad *N* de la solución actual *s*, que denotaremos *N*(*s*)
- Es deseable que el operador de vecindad haga cambios "suaves", de tal forma que cualquier solución de *N(s)* esté relativamente cerca de *s.* El operador de vecindad puede ser, por tanto, similar a un operador de mutación de un algoritmo genético en este sentido

# Introducción
## Estructura de entorno o vecindad

**Ejemplo:**

Solución actual: $s$ = (0 1 1 0 0 1 0 1 1 1)

Si definimos un movimiento como un único cambio de una de sus posiciones:

Tamaño de la vecindad: $|N(s)|$ = 10

Vecinos:

(1 1 1 0 0 1 0 1 1 1)
(0 1 0 0 0 1 0 1 1 1)
(0 1 1 0 1 1 0 1 1 1)
(0 1 1 0 0 1 1 1 1 1)
(0 1 1 0 0 1 0 1 0 1)

(0 0 1 0 0 1 0 1 1 1)
(0 1 1 1 0 1 0 1 1 1)
(0 1 1 0 0 0 0 1 1 1)
(0 1 1 0 0 1 0 0 1 1)
(0 1 1 0 0 1 0 1 1 0)

# Introducción
## Estructura de entorno o vecindad

- **Definición:** En un problema de minimización, una solución *s* se denomina óptimo local con respecto a un operador de vecindad *N*, si $f(s) \leq f(r)$ para todo *r* perteneciente a $N(s)$
- Cuanto más grande sea la vecindad, más costoso será explorarla pero mejor será la calidad de sus óptimos locales
- La clave de la búsqueda local está en encontrar un operador de vecindad eficiente que consiga un buen equilibrio entre la calidad de las soluciones consideradas y el tamaño de la vecindad
- Es deseable que, a partir de una solución cualquiera, se pueda trazar un camino desde esa solución a la solución óptima, mediante sucesivas aplicaciones del operador de vecindad

# Introducción
## Elementos de un algoritmo de búsqueda local

Además de un esquema de representación y una función de fitness necesitamos definir los siguientes elementos:

- Generación de la solución inicial: es decir, un método para obtener la solución inicial. Opciones: aleatoria o heurística (por ejemplo mediante un algoritmo voraz)
- Operador de vecindad: un método para generar soluciones vecinas a la solución actual
- Estrategia de búsqueda en la vecindad: un método para definir cómo se explora la vecindad. Opciones: aleatoria o sistemática
- Criterio de selección/aceptación de vecinos: un método para decidir qué vecino será la siguiente solución actual. Opciones: mejor vecino de la vecindad, el primer vecino que mejore, aleatorio, aceptar un vecino con una probabilidad que dependa de su calidad, …
- Criterio de parada: método para decidir cuándo acaba el algoritmo. Opciones: no hay vecinos que mejoran, tiempo límite, número límite de iteraciones, número límite de iteraciones sin mejora

# Búsqueda local
## Escalada de máximo gradiente vs escalada simple

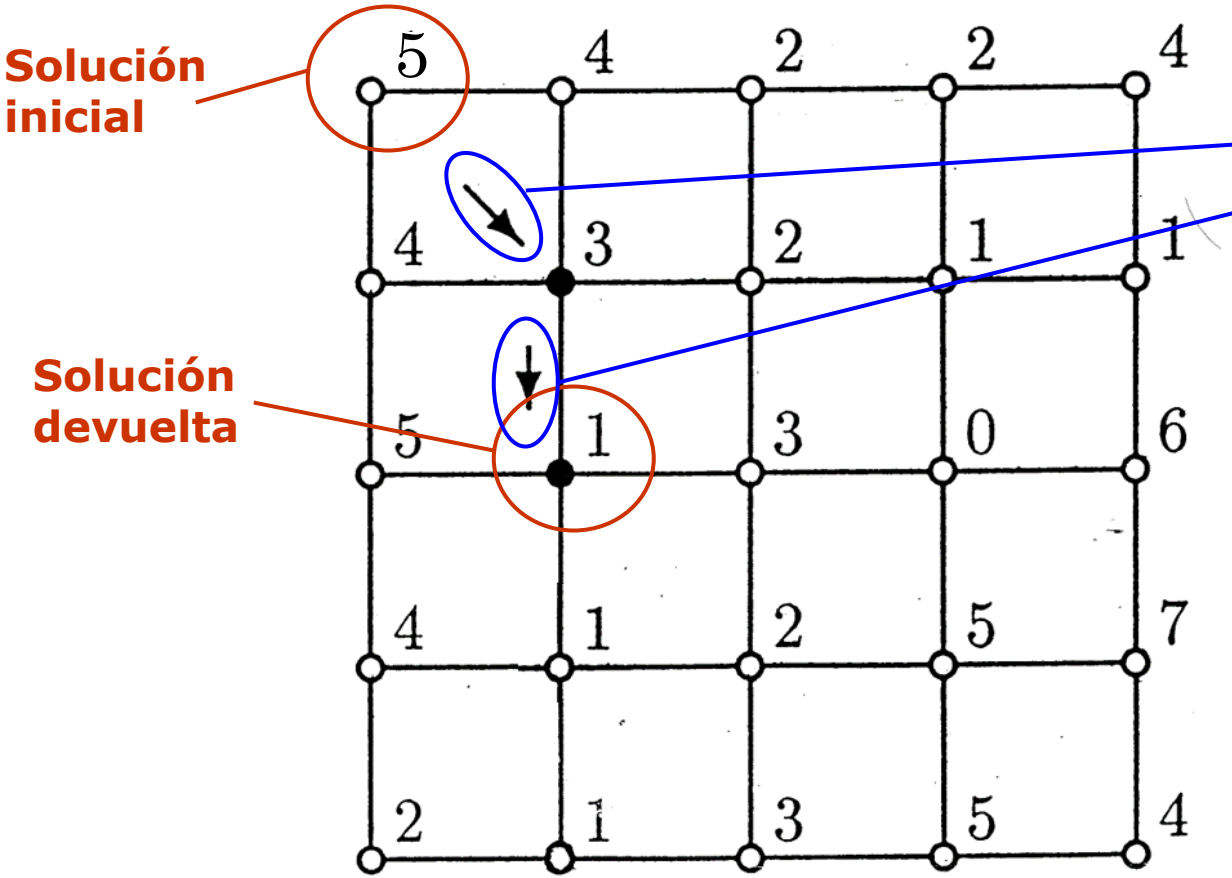
- Los métodos básicos de búsqueda local se basan en examinar las soluciones de la vecindad de la solución actual que mejoren su calidad
- **Escalada de máximo gradiente**
  - Estrategia de búsqueda en la vecindad: Se genera TODA la vecindad de la solución actual
  - Criterio de aceptación: Se escoge el MEJOR vecino, que sustituye al actual solo si mejora
  - Criterio de parada: El algoritmo finaliza cuando ningún vecino mejora
- **Escalada simple**
  - Estrategia de búsqueda en la vecindad: Las soluciones vecinas se generan UNA A UNA de manera sistemática hasta que o bien se encuentra un vecino que mejora o bien se genera toda la vecindad
  - Criterio de aceptación: El PRIMER vecino que mejora que se encuentre sustituye a la solución actual
  - Criterio de parada: El algoritmo finaliza cuando ningún vecino mejora

# Búsqueda local
## Escalada de máximo gradiente

**Procedimiento Escalada de Máximo Gradiente**

**Inicio**
  Solución Inicial ← GENERA();
  Solución Actual ← Solución Inicial;
  **Repetir**
    Hay Mejora ← FALSE;
    Lista de Vecinos ← GENERAR_TODOS_LOS_VECINOS(Solución Actual);
    Mejor Vecino ← SELECCIONAR_MEJOR(Lista de Vecinos);
    **Si** Objetivo(Mejor Vecino) **es mejor que** Objetivo(Solución Actual) entonces
      Solución Actual ← Mejor Vecino;
      Hay Mejora ← TRUE;
  **hasta** (Hay Mejora = FALSE);
  **Devolver**(Solución Actual);
**Fin**

# Búsqueda local
## Escalada de máximo gradiente: Ejemplo

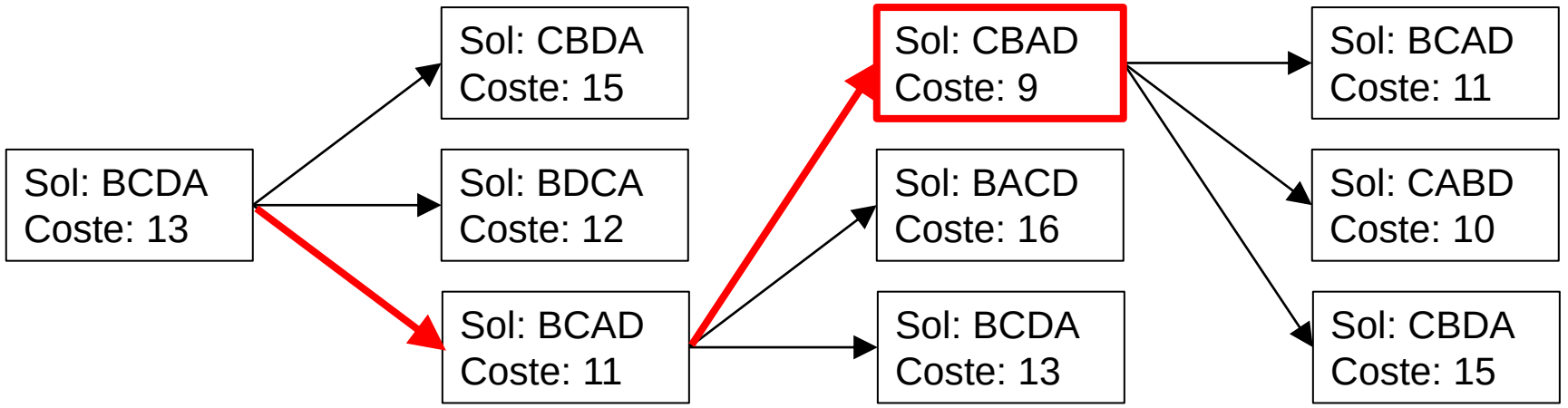



- La vecindad en este ejemplo consiste en moverse a una casilla adyacente, ya sea horizontal vertical o diagonal
- Como vemos, no llega al máximo global, porque tendría que ir a través de una ruta que no implica escoger siempre el mejor camino

VECINDAD: $N(s) = \{s_i \,/\, s_i = (x_i \pm \{0,1\}, y_i \pm \{0,1\}) \wedge s_i \neq s\}$

# Búsqueda local
## Escalada de máximo gradiente: Otro ejemplo

- Problema de minimización (por ejemplo el Traveling Salesman Problem)
- Representación basada en permutaciones
- Vecindad: intercambiar dos posiciones adyacentes




- Se devuelve la solución CBAD, con coste 9, ya que todos sus vecinos son peores

# Búsqueda local
## Escalada simple

**Procedimiento Escalada Simple**

**Inicio**

Solución Inicial ← GENERA();

Solución Actual ← Solución Inicial;

**Repetir**

Hay Mejora ← FALSE;

**Repetir**

Vecino ← GENERAR_UN_VECINO(Solución Actual);

**hasta** (Objetivo(Vecino) es mejor que Objetivo(Solución Actual) **o bien** ya hayamos generado todos los vecinos de Solución Actual);

**Si** Objetivo(Vecino) **es mejor que** Objetivo(Solución Actual) entonces

Solución Actual ← Vecino;

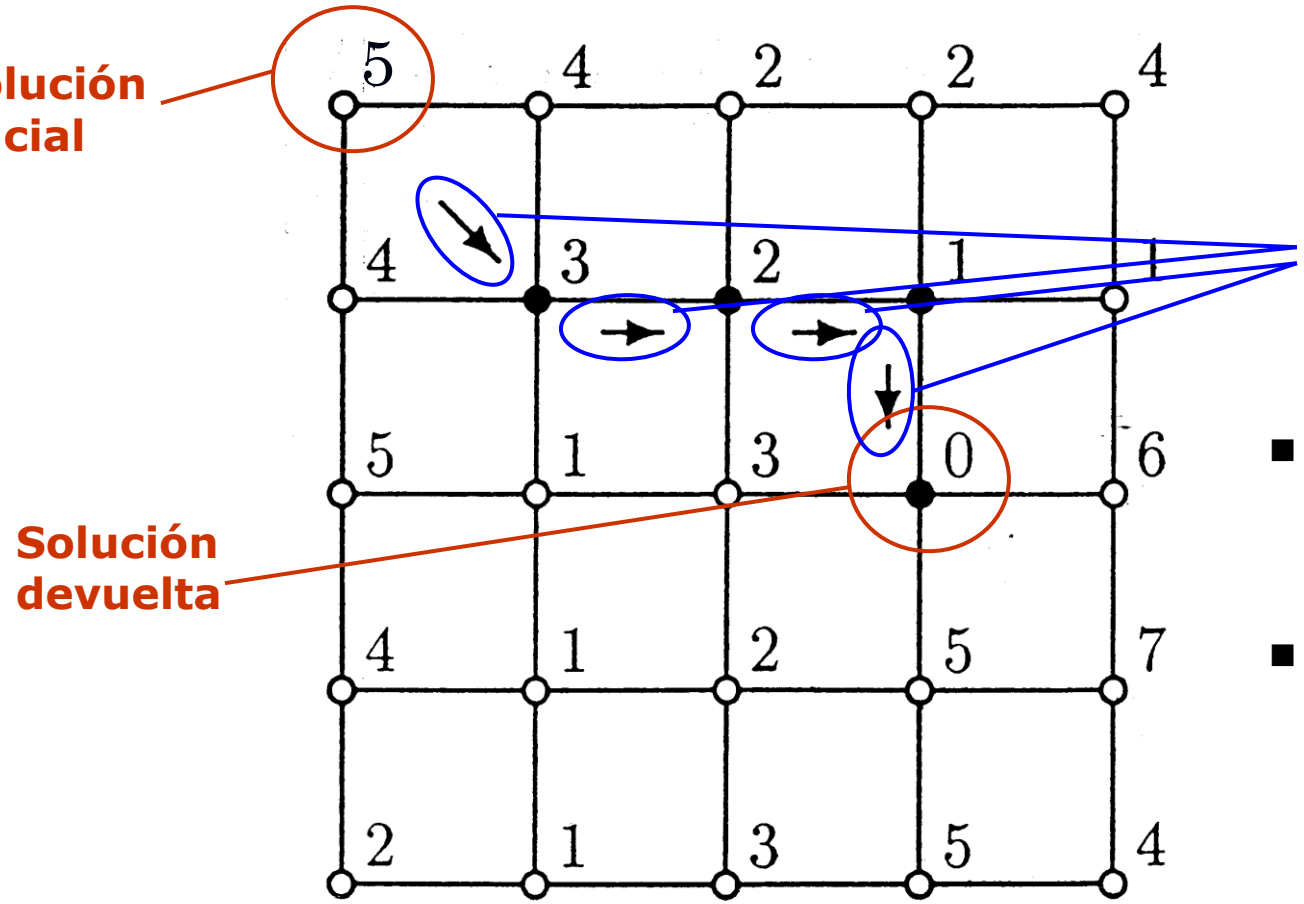
Hay Mejora ← TRUE;

**hasta** (Hay Mejora = FALSE);

**Devolver**(Solución Actual);

**Fin**

# Búsqueda local
## Escalada simple: Ejemplo




**Una posible trayectoria seguida por un método de escalada simple (problema de minimización)**

- En este ejemplo supongamos que los vecinos se generan en un orden aleatorio
- Nótese que dicho orden importa, ya que la trayectoria seguida y la solución final devuelta dependen de ese orden

VECINDAD: $N(s) = \{s_i \,/\, s_i = (x_i \pm \{0,1\}, y_i \pm \{0,1\}) \wedge s_i \neq s\}$

# Búsqueda local
## Escalada de máximo gradiente vs escalada simple

- Escalada de máximo gradiente
  - Converge rápidamente a un óptimo local
  - Posibilidad de que haya convergencia prematura
  - Es muy sensible a la solución inicial
    - (mejores resultados comenzando desde soluciones iniciales heurísticas)
- Escalada simple
  - Menos costosa computacionalmente en vecindades grandes
  - Es muy sensible al orden en el que se explora la vecindad
  - En el peor de los casos debe explorar toda la vecindad
- En muchas aplicaciones ambos métodos consiguen resultados similares

# Ejemplos de operadores de vecindad

- Operadores de vecindad para representación basada permutaciones:
  - Intercambio: elegir dos elementos e intercambiar sus valores
  - Inserción: un elemento de una posición se extrae y se inserta en otra posición
  - Inserción de un bloque: se elige un bloque, se extrae y se inserta en otra posición
- Operadores de vecindad para representación binaria:
  - Inversión de un elemento: se elige un elemento y se cambia su valor
  - Intercambio: elegir dos elementos (cuyo valor sea distinto) e intercambiar sus valores
- Operadores de vecindad para representación entera/real:
  - Modificar un valor: elegir un elemento y asignarle un nuevo valor "cercano" al actual
  - Inserción: un elemento de una posición se extrae y se inserta en otra posición
- Nótese que muchos de estos operadores de vecindad son similares a los operadores de mutación de los algoritmos genéticos (hacen cambios pequeños)
- A veces puede ser interesante reducir el tamaño de la vecindad. Por ejemplo en el operador de intercambio es razonable no considerar cualquier intercambio, sino solo intercambios entre posiciones cercanas, para reducir el coste computacional

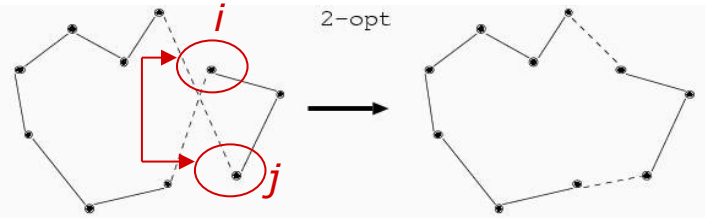
# Otras consideraciones más avanzadas
## Evaluación incremental de la calidad de los vecinos

- La evaluación de la función objetivo suele ser la parte más costosa computacionalmente de un método de búsqueda local, y en general de cualquier metaheurística.
- Una opción para acelerar esto es hacer una evaluación aproximada de los vecinos, y solo hacer una evaluación completa del vecino en caso de elegirlo
- Otras veces podemos aprovechar que los cambios que hacemos a la solución para obtener un vecino son pequeños, y así calcular su coste de una forma más eficiente
- Podemos evaluar el coste del movimiento $\Delta(s,m)$ en lugar de calcular el coste de la solución completa, es decir, hacer $s'=s \oplus m$. $\Delta(s,m)$ representa la diferencia de coste entre $f(s)$ y $f(s')$
- Por tanto, podríamos calcular rápidamente el coste del vecino: $f(s') = f(s) - \Delta(s,m)$
- La complejidad de definir esta evaluación incremental depende del operador de vecindad y del problema en particular

# Otras consideraciones más avanzadas
## Evaluación incremental de la calidad de los vecinos

**EJEMPLO:** Evaluación incremental del operador de vecindad de intercambio para el Problema del Viajante de Comercio

- Como únicamente se intercambia la posición de dos ciudades *i* y *j* en la permutación, hasta cuatro arcos como máximo podrían ser diferentes entre las dos soluciones




- La diferencia de coste $\Delta(s,i,j)$ entre ambas soluciones sería:

$$D(s[i\text{-}1],s[j]) + D(s[j],s[i\text{+}1]) + D(s[j\text{-}1],S[i]) + D(s[i],s[j\text{+}1])$$
$$- D(s[i\text{-}1],s[i]) - D(s[i],s[i\text{+}1]) - D(s[j\text{-}1],s[j]) - D(s[j],s[j\text{+}1])$$

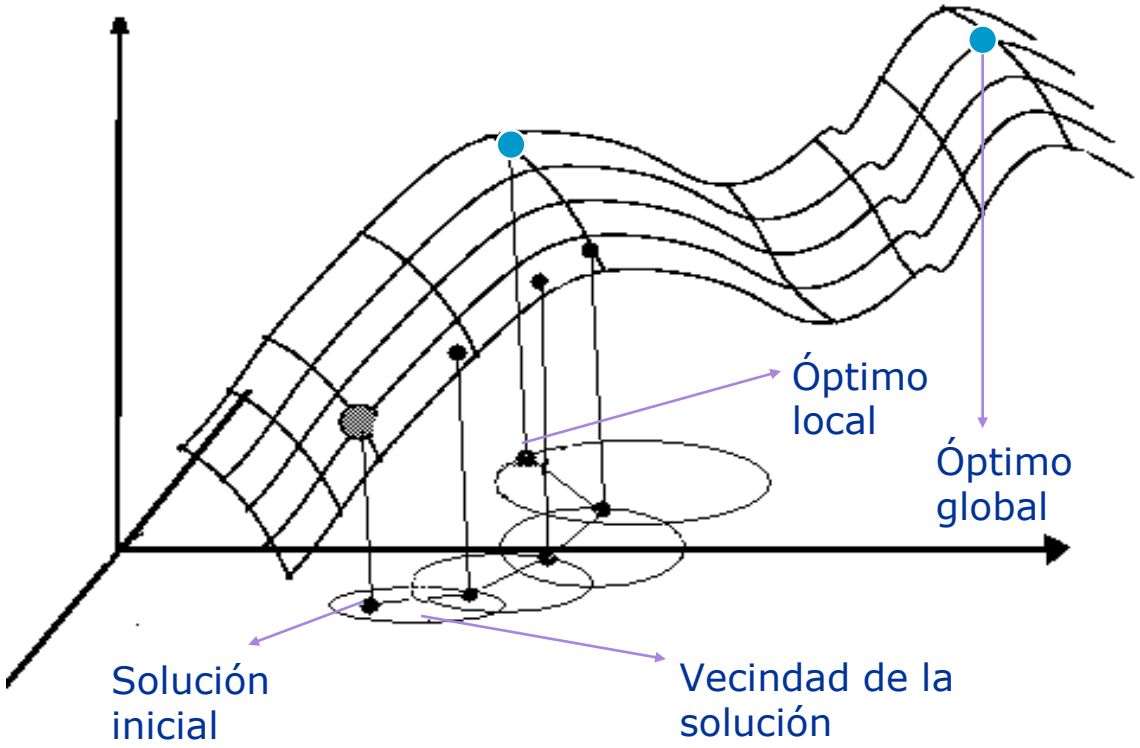
- Por tanto, evaluando incrementalmente la solución vecina tendríamos que calcular como mucho 8 valores, en lugar de los *n* requeridos en una evaluación completa

# Otras consideraciones más avanzadas
## Estrategias avanzadas para explorar la vecindad

- Cuando el tamaño de la vecindad es muy grande (exponencial con el tamaño del problema), explorarla entera puede ser muy costoso. Una exploración parcial heurística puede ser recomendable para reducir los tiempos de ejecución
- En estos casos, y en cualquier caso en el que el criterio de aceptación se base en escalada simple, puede ser interesante utilizar información heurística específica del problema para:
  1. Restringir la vecindad de una solución de tal forma que solo incluya movimientos prometedores
  2. Definir un orden sistemático de exploración de la vecindad que aplique en primer lugar los movimientos más prometedores (por ejemplo en el TSP tratar de quitar algún arco muy largo)

# Métodos de búsqueda local básicos
## Principal problema

- Como consecuencia de elegir solo movimientos que mejoran, los métodos de búsqueda local básicos suelen converger a óptimos locales y quedarse atrapados en ellos, y esos óptimos locales podrían estar muy lejos del óptimo global



# Métodos de búsqueda local básicos
## ¿Cómo escapar de óptimos locales?

Hay diferentes opciones para diseñar un algoritmo de búsqueda local que sea capaz de escapar de óptimos locales:

- Aceptar vecinos que no mejoran, o incluso empeoran, a la solución actual: método utilizado en enfriamiento simulado y búsqueda tabú

- Volver a comenzar la búsqueda desde otra solución inicial: método utilizado por búsqueda multiarranque, iterated local search, GRASP, …

- Modificar la estructura de vecindad durante la búsqueda: método utilizado por búsqueda en entornos variables

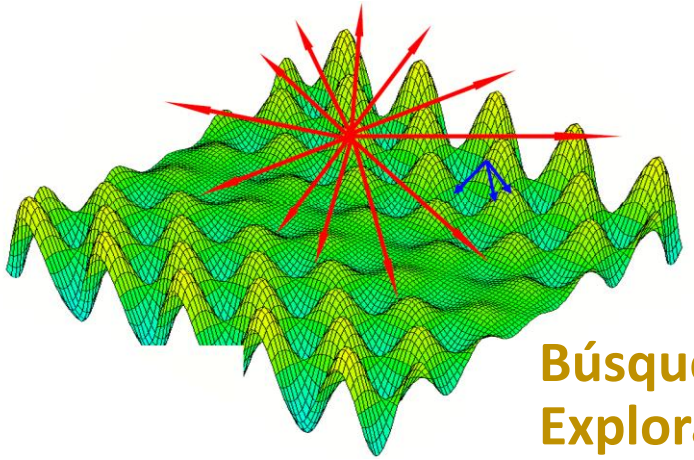
- Modificar la función objetivo: método utilizado por guided local search, smoothing strategies, y noising methods

# Contenidos

1. Metaheurísticas: introducción y clasificación
2. Algoritmos genéticos
3. Introducción a la búsqueda local
4. **Algoritmos meméticos**
5. Introducción a la optimización multiobjetivo
6. Conclusiones

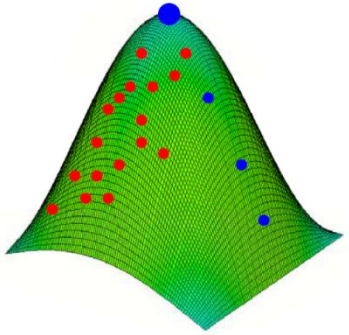
# Algoritmos híbridos
## Poblaciones vs trayectorias

- Los algoritmos evolutivos son buenos exploradores
- Los algoritmos de búsqueda local son malos exploradores



**Búsqueda global**
**Exploración**

- Los algoritmos evolutivos son malos explotadores
- Los algoritmos de búsqueda local son buenos explotadores



**Búsqueda local**
**Explotación**

- **Algoritmo Memético** = Algoritmo Evolutivo + Búsqueda Local, y por tanto combina la capacidad de EXPLORACIÓN de un algoritmo evolutivo con la capacidad de EXPLOTACIÓN de una búsqueda local

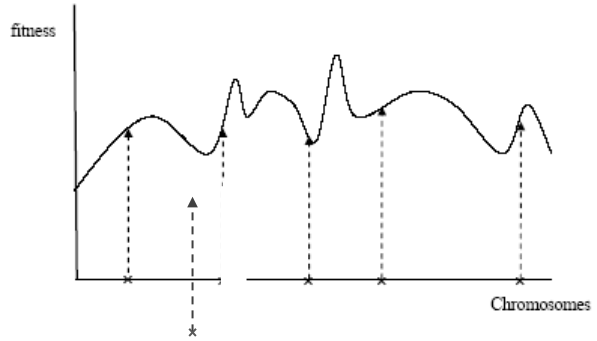
# Algoritmos meméticos
## Consideraciones sobre la aplicación de la BL

- ¿Cuándo y cómo debemos aplicar la búsqueda local dentro del ciclo evolutivo? ¿Qué individuos de la población debemos mejorar y cómo debemos elegirlos? ¿Cuánto tiempo de computación debemos dedicar a cada búsqueda local?
- Algunas posibilidades:
  - Aplicar búsqueda local a todos los cromosomas, es decir a todos los individuos de la población inicial y a todos los nuevos individuos que se generen a lo largo de la ejecución (en este caso deberíamos dedicar muy poco tiempo de computación a cada búsqueda local)
  - Aplicar búsqueda local a cada individuo con una determinada probabilidad
  - Aplicar búsqueda local solo a los *n* mejores individuos en cada generación
  - Aplicar búsqueda local únicamente a la solución final devuelta por el genético
- Uno de los factores que condiciona esta decision es el coste computacional de la búsqueda local en nuestro problema. Solo si es una búsqueda local poco costosa podremos optar por aplicarla a todos los cromosomas

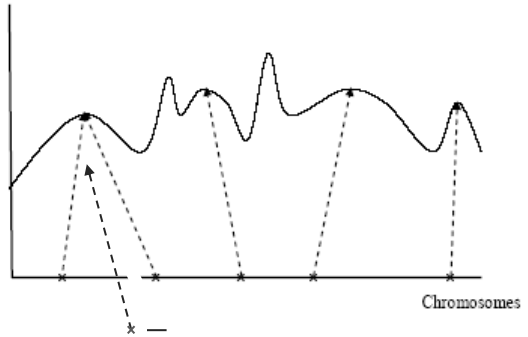
# Algoritmos meméticos
## Evolución Baldwiniana vs Lamarckiana

- Hay dos modelos evolutivos que podemos utilizar al combinar una búsqueda local con un algoritmo genético:
- Evolución Baldwiniana
  - El proceso de asignar a un cromosoma el valor de fitness de la solución mejorada alcanzada desde el a través de búsqueda local se conoce como efecto Baldwin
  - Es esperable que las características aprendidas se vayan asimilando gradualmente por los individuos en posteriores generaciones
- Evolución Lamarckiana
  - La evolución Lamarckiana asume que todo lo que el individuo aprende durante su vida se codifica de nuevo en el cromosoma
  - Su implementación en un algoritmo memético es: la solución alcanzada por la búsqueda local se codifica en un nuevo cromosoma que sustituye al original en la población
- Habitualmente la evolución Lamarckiana da mejores resultados

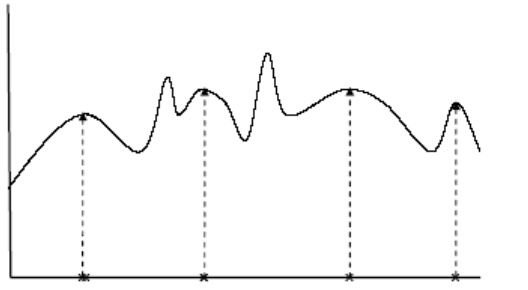
# Algoritmos meméticos
## Evolución Baldwiniana vs Lamarckiana




a) Fitness landscape and fitness values in a genetic algorithm




b) With Baldwinian evolution fitness values correspond to local optima reached by local search




c) With Lamarckian evolution chromosomes are transformed accordingly to fitness values from local search.

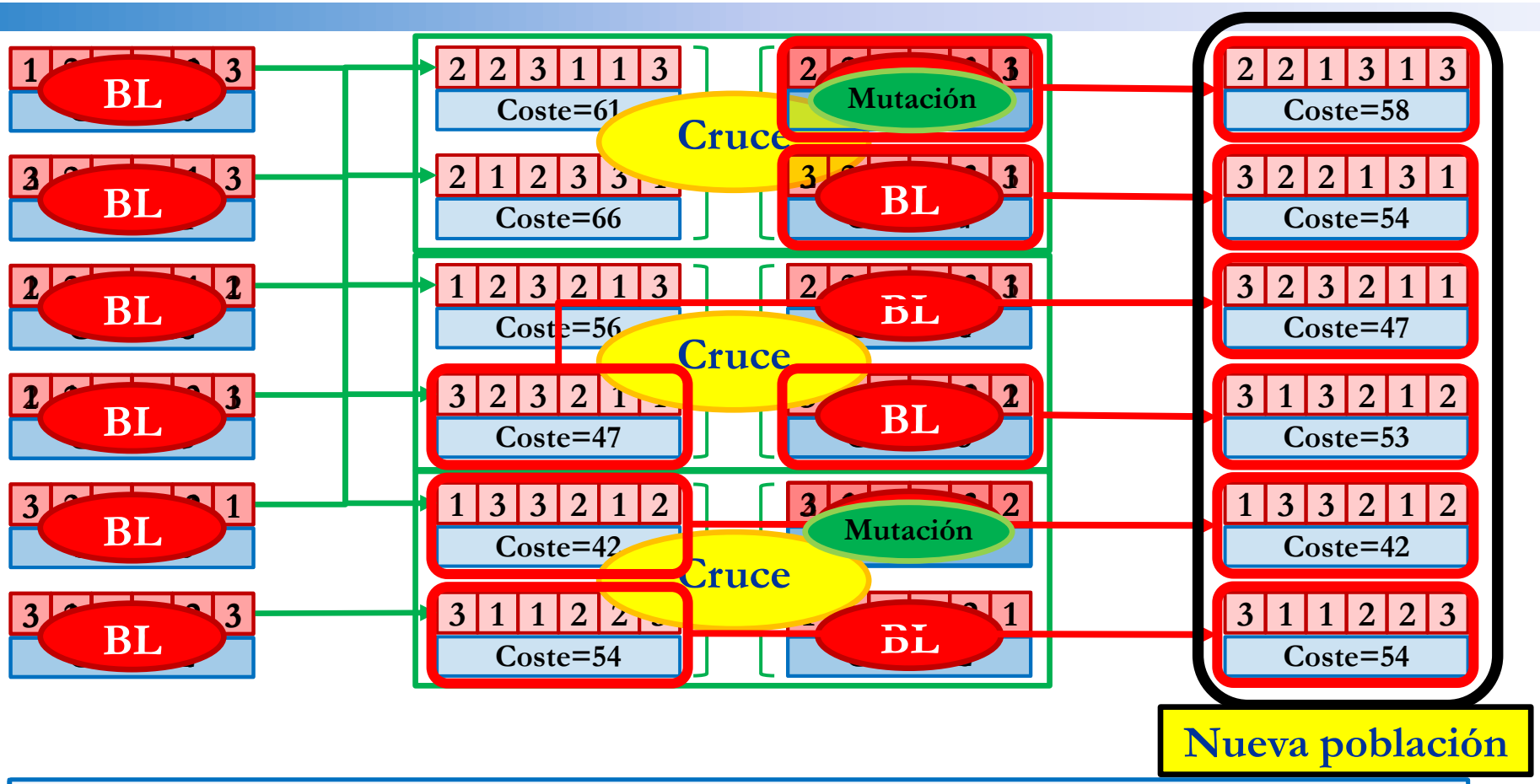
**Figure 1.** Differences among genetic and memetic algorithms with Baldwinian and Lamarckian evolution

Un algoritmo genético busca en todo el espacio de soluciones

Un algoritmo memético busca en un subespacio de óptimos locales

# Algoritmos meméticos

## Ejemplo de ejecución




**8) Repetir 3-7 hasta criterio de parada**

# Contenidos

1. Metaheurísticas: introducción y clasificación
2. Algoritmos genéticos
3. Introducción a la búsqueda local
4. Algoritmos meméticos
5. **Introducción a la optimización multiobjetivo**
   - **Descripción**
   - Algoritmos genéticos multiobjetivo
   - Métricas para algoritmos multiobjetivo
6. Conclusiones

# Optimización multiobjetivo
## Introducción

- Muchos problemas requieren optimizar varios objetivos simultáneamente
- Por ejemplo la mayor parte de los problemas en ingeniería de diseño requieren minimizar costes y maximizar el rendimiento y fiabilidad
- El coste, el rendimiento y la fiabilidad son a menudo objetivos contradictorios: para reducir los costes tenemos que reducir la fiabilidad o el rendimiento
- La optimización mono-objetivo es un caso particular de optimización multiobjetivo

# Optimización multiobjetivo
## Formulación del problema

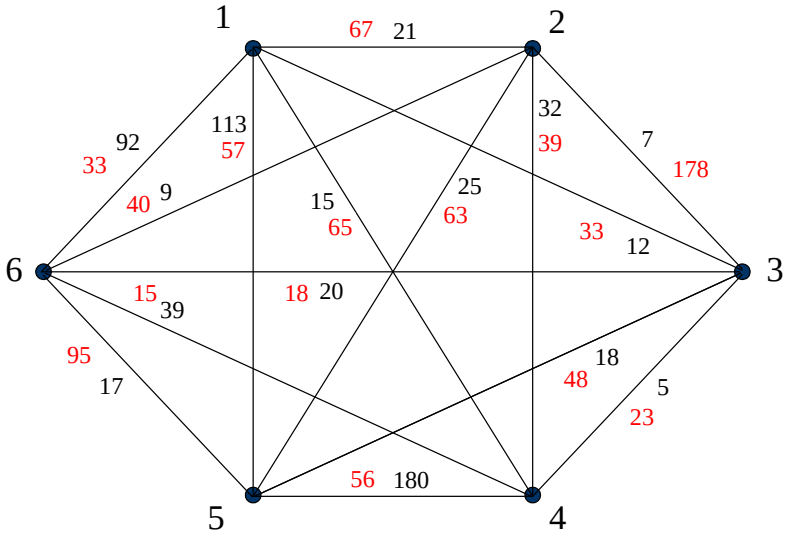
- Dado un vector n-dimensional de variables de decisión $x=\{x_1,\ldots,x_n\} \in \mathbf{X}$, encontrar un vector $x^*$ que minimice (o maximice) un conjunto de K funciones objetivo $z(x) = \{z_1(x),\ldots,z_K(x)\} \in \mathbf{Y}$
  - **X** es el espacio de decisión
  - **Y** es el espacio objetivo. Generalmente $\mathbf{Y} \subseteq R^K$
  - $\{z_1,\ldots,z_K\}$ es el conjunto de funciones objetivo
  - Puede haber restricciones adicionales:
    - Desigualdades
    - Igualdades
    - Otras

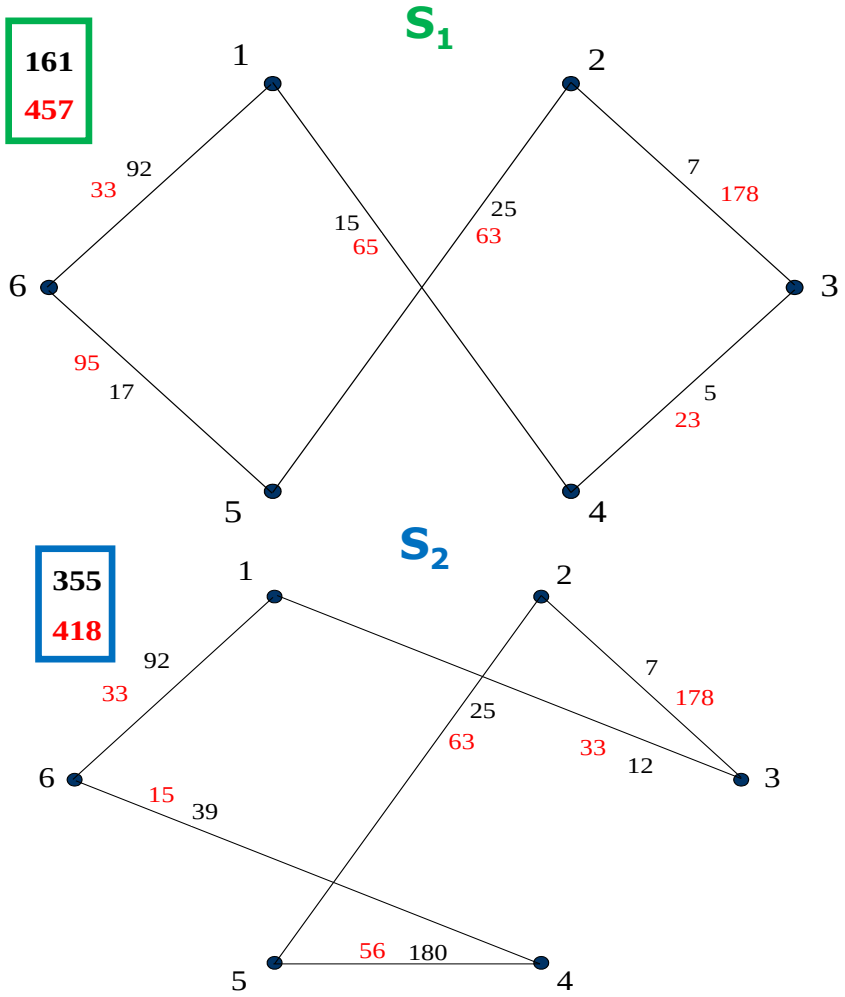
# Optimización multiobjetivo
## Un ejemplo: TSP con distancia y coste

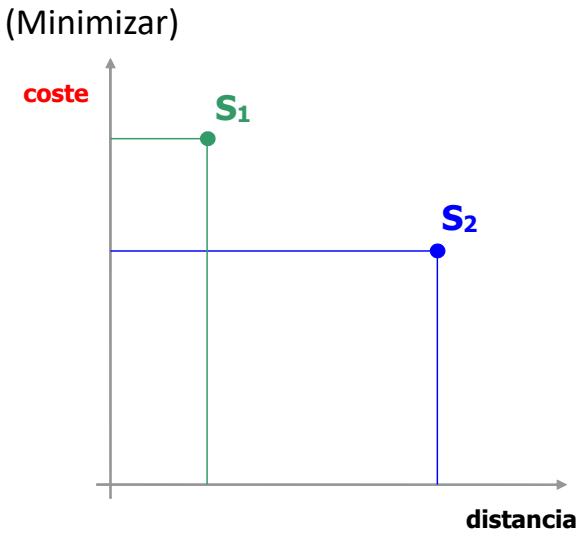
- Ejemplo: TSP con distancias y costes en cada arco
  - Podemos imaginar que utilizamos autopistas de peaje, y entonces puede ocurrir que caminos más cortos sean más caros, por lo que distancia y coste podrían ser objetivos contradictorios
  - $\mathbf{X} = \mathbf{C}^{\mathbf{n}}$,
    - **C** es el conjunto de ciudades
    - **n** es el número de ciudades
  - $\mathbf{Y} \subseteq R^2$
  - Funciones objetivo
    - $z_1(x)$ = distancia de la ruta x
    - $z_2(x)$ = coste necesario para recorrer la ruta x
  - Restricciones
    - $x_i \neq x_j$;  $1 \leq i,j \leq n$; $i \neq j$  (es decir, que cada ciudad se visita una única vez)

# Optimización multiobjetivo

## Un ejemplo: TSP con distancia y coste



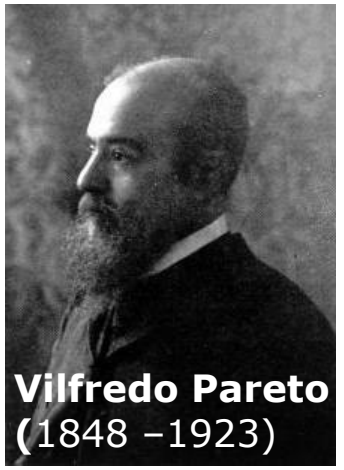
# Optimización multiobjetivo
## Tres enfoques

- Agregación de funciones
  - Todos los objetivos se agregan en uno único, por ejemplo ponderando cada uno de ellos mediante pesos. Y luego se aplica una metaheurística mono-objetivo
    - $f(x) = F(z_1(x),\ldots,z_K(x))$ (por ejemplo $z_1w_1+z_2w_2+\ldots+z_kw_k$)
    - Debemos tener cuidado con la magnitud de los objetivos, podría ser necesario escalar el fitness
- Orden lexicográfico
  - Los objetivos se consideran en un orden jerárquico
    - Primero min $z_{[1]}(x)$, después min $z_{[2]}(x)$, después min $z_{[3]}(x)$, …
    - Es decir, le damos más importancia al primer objetivo y utilizamos los demás para desempatar entre soluciones cuyo valor del primer objetivo sea idéntico
- Optimización Pareto
  - El objetivo es encontrar el frente Pareto, o al menos una buena aproximación

# Optimización multiobjetivo
## Conceptos de dominancia

*(la siguiente definición sería para problemas de minimización)*

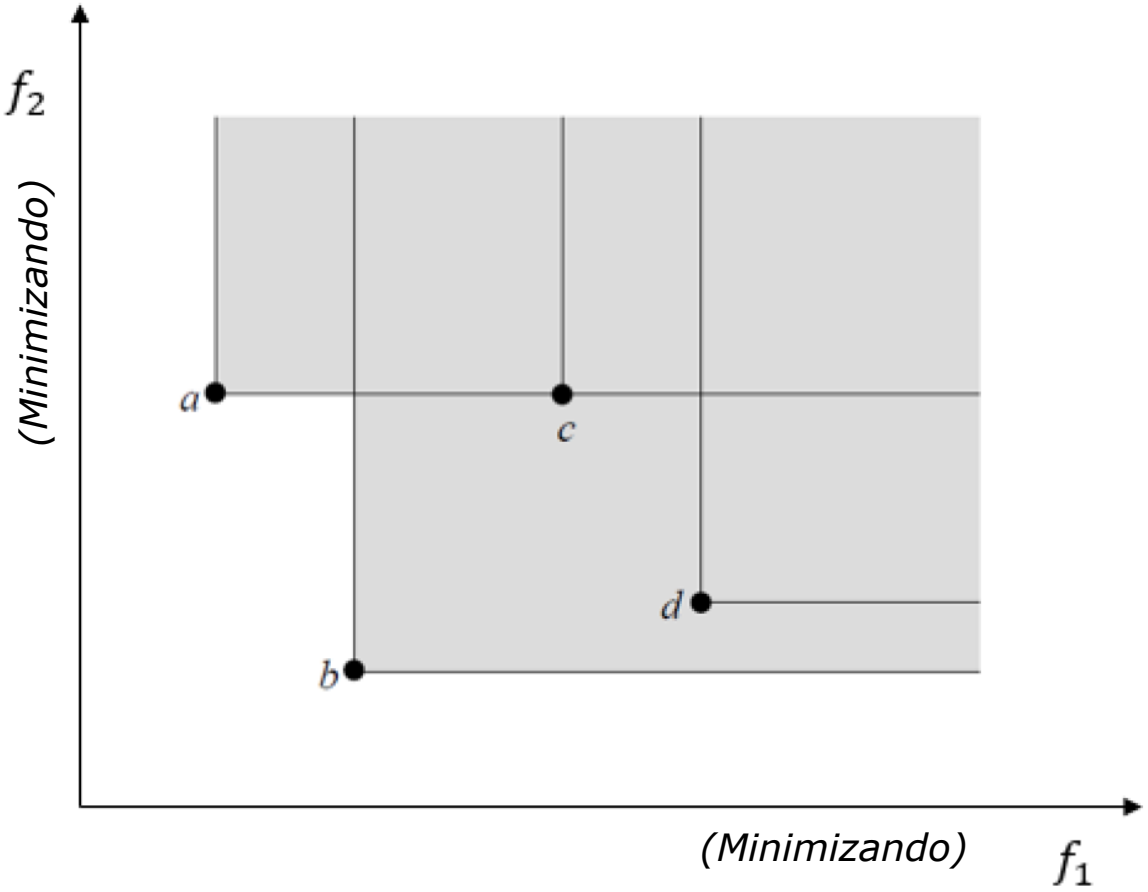
- Dadas dos soluciones x e y, x domina y (se denota $x \succ y$), si y solo si $z_i(x) \leq z_i(y)$ para todo $1 \leq i \leq K$, y además $z_j(x) < z_j(y)$ para al menos un valor de j (en términos generales denotamos por $z(x) \succ z(y)$ que z(x) domina a z(y))
  - Es decir, que una solución domina a otra si es mejor o igual en todos los objetivos, y estrictamente mejor en al menos uno de ellos
- Una solución es Pareto óptima si no está dominada por ninguna otra en el espacio solución
- El conjunto de todas las soluciones no dominadas $X^* \subseteq \mathbf{X}$ es el Conjunto Pareto óptimo y representa la solución óptima del problema multiobjetivo
- Los valores de la función objetivo del conjunto Pareto óptimo $z(X^*) \subseteq \mathbf{Y}$ forman el Frente Pareto



# Optimización multiobjetivo
## Conceptos de dominancia: ejemplo

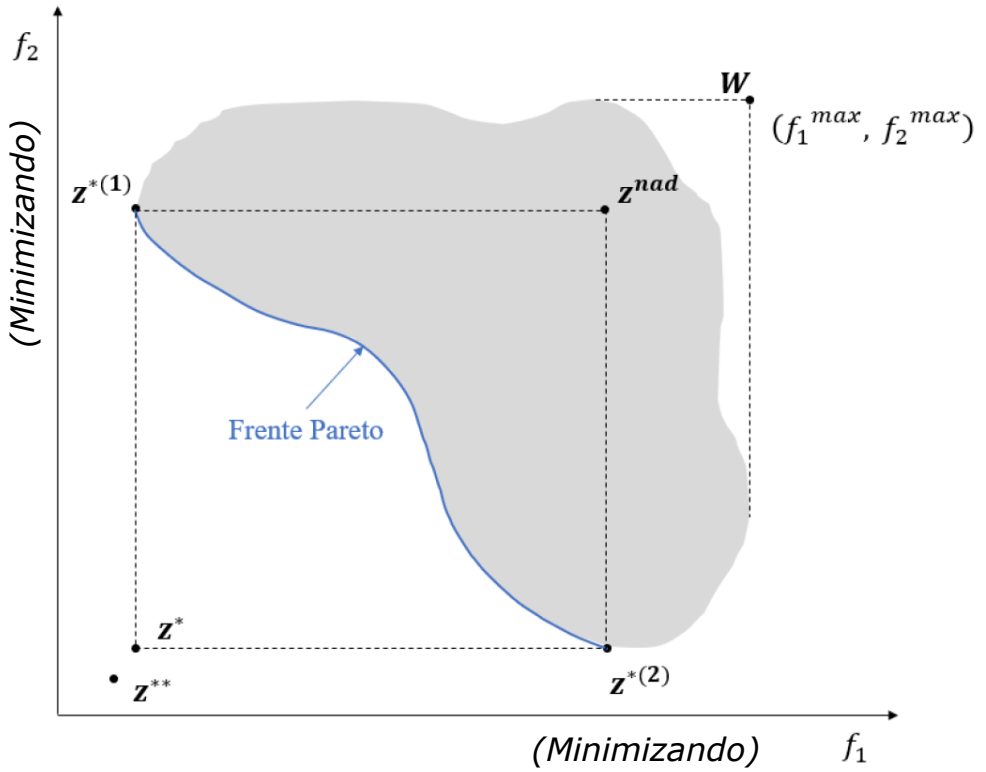
*Consideremos las soluciones a, b, c, d*

- $a \succ c$ : *a* domina a *c* porque es mejor en el objetivo $f_1$ e igual en el objetivo $f_2$
- *a* no domina ni a *b* ni a *d*, porque, aunque es mejor en el objetivo $f_1$, es peor en el objetivo $f_2$
- $b \succ c$ y $b \succ d$ : *b* domina a *c* y a *d* porque es mejor que ellas en ambos objetivos
- Las soluciones *c* y *d* no dominan a nadie



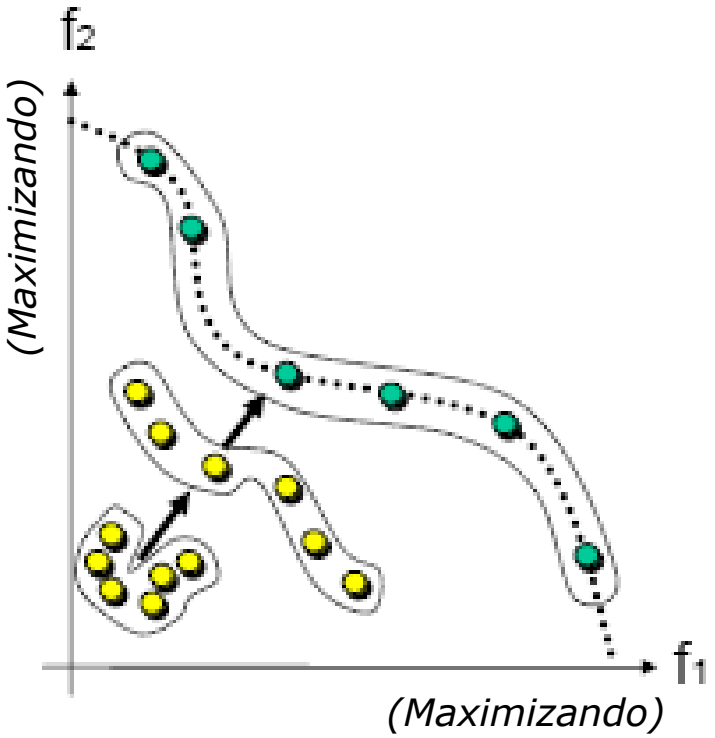
# Optimización Pareto
## Vectores característicos

- Vector objetivo ideal $z^*$: vector con el valor óptimo de cada objetivo
  - Normalmente es inalcanzable (a no ser que los objetivos no sean conflictivos entre sí)
  - Se utiliza como punto de referencia o para normalizar
- Vector objetivo utópico $z^{**}$: vector con valores ligeramente mejores a los del vector objetivo ideal
  - A veces se requiere una solución de comparación cuyo valor objetivo sea estrictamente mejor a la de cualquier solución en el espacio de búsqueda
- Vector objetivo nadir $z^{nad}$: vector con el peor valor de cada objetivo en el frente Pareto
  - Se utiliza para normalizar
- Vector de máximos $W$: vector con el peor valor de cada objetivo en todo el espacio de búsqueda



# Optimización Pareto
## Objetivo

- En la optimización Pareto el objetivo es encontrar una aproximación al frente Pareto que se aproxime al verdadero frente Pareto tanto como se pueda
  - Debe estar lo más cerca posible del verdadero frente Pareto
  - Las soluciones deben distribuirse uniformemente y ser distintas en el frente Pareto
  - Debe capturar todo el espectro del frente Pareto, incluyendo los extremos finales




Solución del Frente Pareto

Solución de una aproximación del frente Pareto

# Contenidos

1. Metaheurísticas: introducción y clasificación
2. Algoritmos genéticos
3. Introducción a la búsqueda local
4. Algoritmos meméticos
5. **Introducción a la optimización multiobjetivo**
   - Descripción
   - **Algoritmos genéticos multiobjetivo**
   - Métricas para algoritmos multiobjetivo
6. Conclusiones

# Algoritmos genéticos multiobjetivo
## Introducción

- Los algoritmos genéticos son muy prometedores como algoritmos de búsqueda multiobjetivo, por varias razones
  - Están basados en poblaciones, por lo que son adecuados para obtener múltiples soluciones
  - Se pueden adaptar para buscar en diferentes regiones del espacio de búsqueda simultáneamente
  - No son sensibles a la forma que tenga el frente Pareto
- La mayoría de las propuestas multiobjetivo en la literatura son metaheurísticas, en particular algoritmos evolutivos (70%)
- La mayoría de ellos (90%) tratan de aproximar el conjunto Pareto

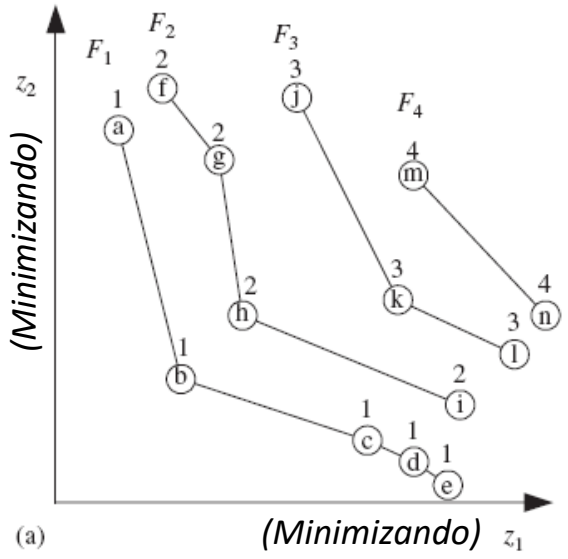
# Algoritmos genéticos multiobjetivo
## Algunos enfoques clásicos *[Abdullah, 2006]*

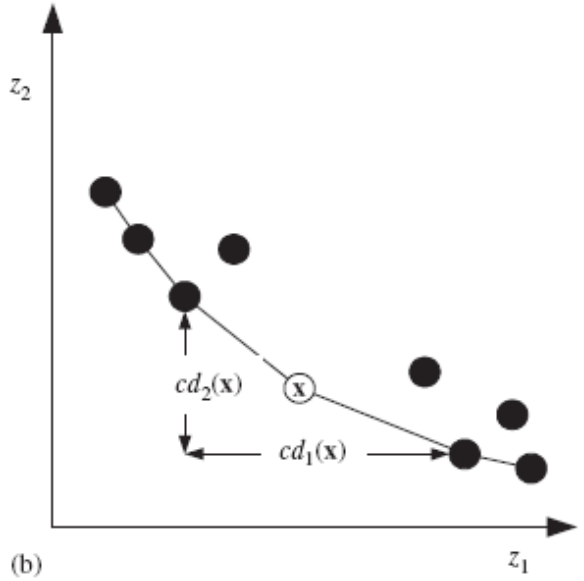
- VEGA: Vector Evaluated GA *[Schaffer, 1985]*
- MOGA: MultiObjective GA *[Fonseca, 1993]*
- WBGA: Weight Based GA *[Hajela, 1992]*
- RWGA: Random Weighted GA *[Murata, 1995]*
- NSGA: Non-dominated Sorting GA *[Srinivas, 1994]*
- NSGA-II: Non-dominated Sorting GA - II *[Deb, 2002]*
- SPEA: Strength Pareto EA *[Zitzler, 1999]*
- SPEA2: Strength Pareto EA 2 *[Zitzler, 2001]*
- PESA: Pareto Envelope-based Selection Algorithm *[Corne, 2000]*
- PAES: Pareto Archived Evolution Strategy *[Knowles, 1999]*
- RDGA: Rank-Density based GA *[Lu, 2003]*
- DMOEA: Dynamic MO EA *[Yen, 2003]*
- NPGA: Niched Pareto GA *[Horn, 1994]*
- …

# NSGA-II *[Deb, 2002]*

## Fitness: Dominance depth y crowding distance

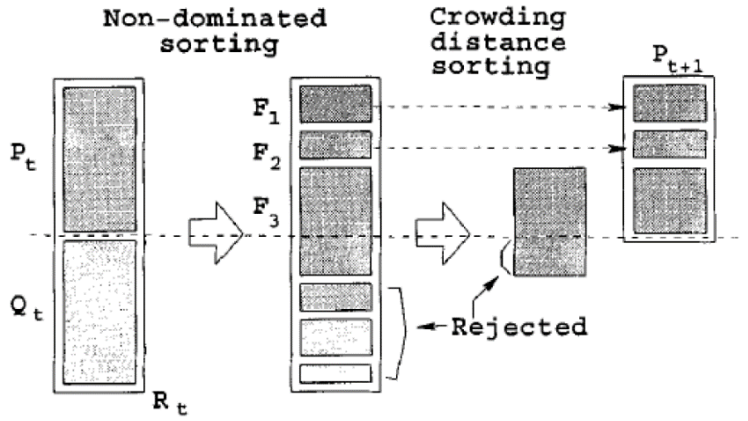
- Dominance Depth: divide la población en una serie $F_1$, …, $F_n$ de frentes no-dominados, donde $F_1$ es el frente Pareto de la población
- Crowding distance: medida de la dispersión local de las soluciones en un frente Pareto $F_i$. El objetivo es obtener una distribución uniforme de las soluciones a lo largo de cada frente Pareto. Se utiliza para desempatar entre soluciones con un mismo dominance Depth en los operadores de selección y reemplazamiento



# NSGA-II *[Deb, 2002]*
## Operadores de selección y reemplazamiento

- La selección y el reemplazamiento son la principal diferencia respecto a un algoritmo genético mono-objetivo
- Selección
  - Torneo binario: seleccionamos al azar dos soluciones *x* e *y*. La ganadora será la que tenga el dominance depth más bajo. En el caso de que estén en el mismo frente, la que tenga mayor distancia de crowding es la ganadora, para mantener diversidad
- Reemplazamiento
  - Mantiene completos todos los primeros i frentes Pareto que quepan en la población, combinando los padres y los descendientes, en la fase de reemplazo: $|F_1|+\ldots+|F_i| < N$
  - Para el resto de individuos se toman aquellos de $F_{i+1}$ con máxima crowding distance
  - De esta forma, todas las soluciones no-dominadas se mantienen en la población si $|F_1| \leq N$
- El NSGA-II es un algoritmo multiobjetivo muy extendido y considerado por muchos el mejor
  - Inconveniente: Problemas exploratorios conforme se incrementa el número de funciones objetivo
    - NSGA-III *[Deb, 2014]* trata de solucionar esto



# Algoritmos meméticos multiobjetivo
## Introducción

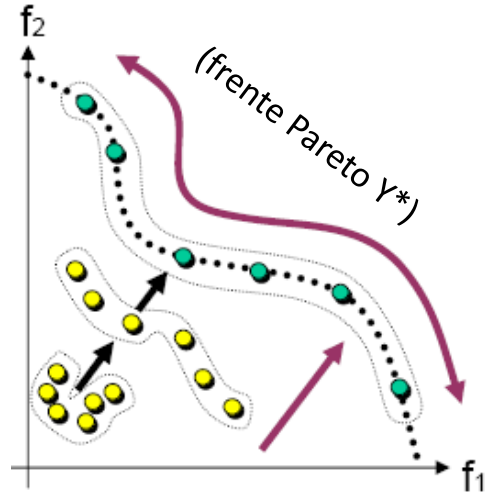
- Frecuentemente se hibrida una búsqueda local con un algoritmo evolutivo, para combinar la buena capacidad de exploración de los algoritmos evolutivos con la buena capacidad de explotación de las búsquedas locales
- Pero, ¿cómo implementar una búsqueda local multiobjetivo?
- El problema principal es: ¿cómo decidir si un vecino mejora o empeora?
  - Si el vecino domina a la solución actual, parece claro que es mejor
  - Pero, ¿y si el vecino es mejor en unos objetivos y peor en otros?
- Algunas propuestas de búsqueda local multiobjetivo en la literatura:
  - PAES (Pareto Archived Evolution Strategy) [Knowles and Corne, 2000]
  - Pareto Local Search [Paquete et al., 2007]
  - HCS (Hill Climber with Sidestep) [Lara et al., 2010]
  - Multi-objective hill climbing local search [González et al., 2016]
  - …

# Contenidos

1. Metaheurísticas: introducción y clasificación
2. Algoritmos genéticos
3. Introducción a la búsqueda local
4. Algoritmos meméticos
5. **Introducción a la optimización multiobjetivo**
   - Descripción
   - Algoritmos genéticos multiobjetivo
   - **Métricas para algoritmos multiobjetivo**
6. Conclusiones

# Métricas para algoritmos multiobjetivo
## Introducción

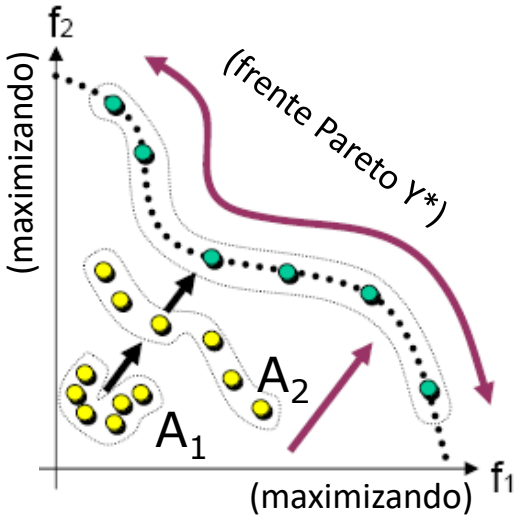
- ¿Cómo podemos evaluar el rendimiento de los algoritmos multiobjetivo?
  - En cuanto a los recursos computacionales
    - Tiempo de ejecución
    - Número de evaluaciones del fitness
  - En cuanto a la calidad de la aproximación del conjunto Pareto
    - Cercanía a Y*
    - Distribución uniforme
    - Cobertura del espacio objetivo
    - …



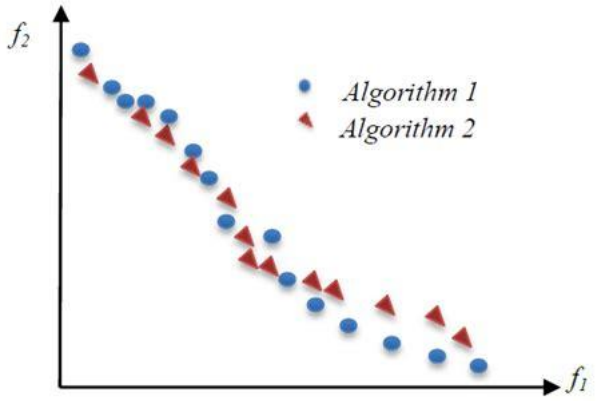
# Métricas para algoritmos multiobjetivo
## Introducción

- ¿Cómo comparamos dos algoritmos multiobjetivo A1 y A2 si consumen recursos similares?
  - $A_2$ es mejor que $A_1$ si la aproximación al frente Pareto $Y_2$ calculada por $A_2$ "domina" a la aproximación al frente Pareto $Y_1$ calculada por $A_1$, es decir:

    $\forall a_1 \in Y_1 \; \exists \; a_2 \in Y_2$ tal que $a_2 \succ a_1$



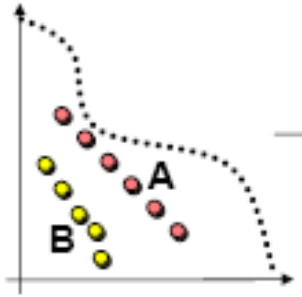

  - Pero si ni $Y_1$ domina a $Y_2$, ni $Y_2$ domina a $Y_1$, en principio ambos algoritmos no son comparables y no se podría decir que uno es mejor que otro



# Métricas para algoritmos multiobjetivo
## Indicadores de calidad

- Indicadores unarios: asignan a cada aproximación un número real I(A)
  - Ejemplos: hipervolumen, distancia generacional, distribución uniforme de soluciones, extensión de la aproximación al frente Pareto, cardinalidad, …
- Indicadores binarios: asignan a cada par de aproximaciones un número real I(A,B)
  - Ejemplos: coverage indicator, binary $\epsilon$–indicator, …

**Example:** unary indicators combined




| | A | B |
|---|---|---|
| hypervolume | 432.34 | 420.13 |
| distance | 0.3308 | 0.4532 |
| diversity | 0.3637 | 0.3463 |
| spread | 0.3622 | 0.3601 |
| cardinality | 6 | 5 |

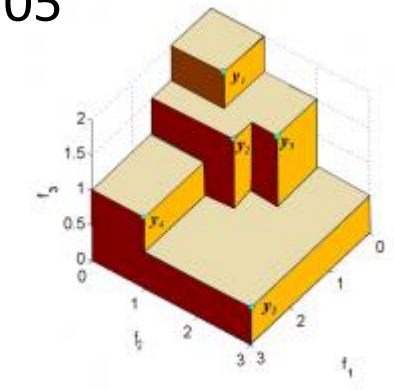
→ "A better"

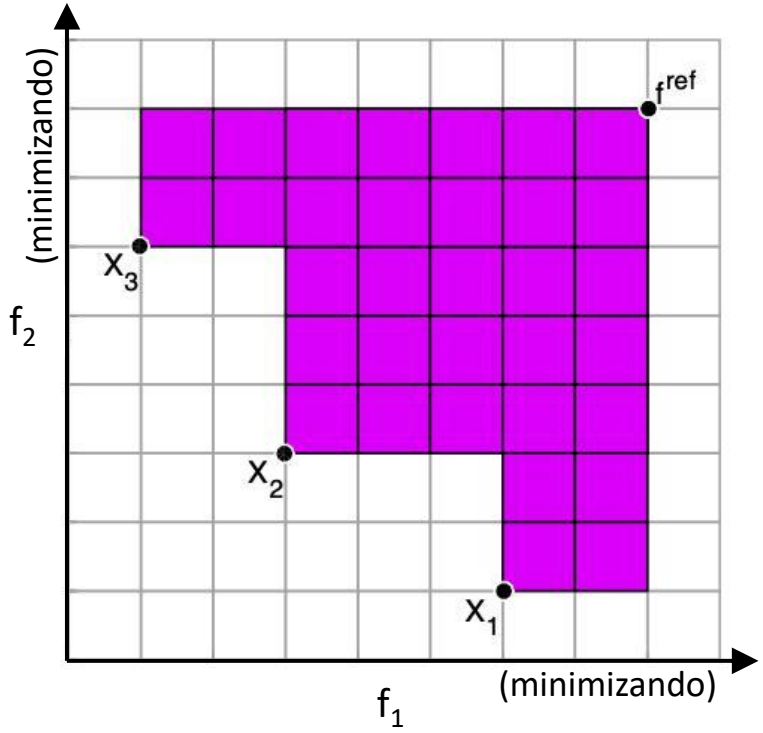
© Eckart Zitzler ETH Zurich Two Decades of EMO 26

- *[Zitzler et al., 2003]*: se necesitaría un número infinito de indicadores unarios para detectar, en general, si A es mejor que B

# Métricas para algoritmos multiobjetivo
## Indicadores unarios: hipervolumen

- Hipervolumen (Hypervolume) *[Zitzler, 1999]*
  - Es posiblemente el indicador más utilizado en la literatura
  - Volumen del espacio de objetivos dominado por la aproximación al frente Pareto *Y*, y delimitado por un punto de referencia (dicho punto de referencia debe ser peor en todos los objetivos)
  - Como punto de referencia, una práctica habitual es utilizar el peor valor de cada objetivo que hayamos encontrado durante nuestras ejecuciones, multiplicado por 1,05
  - Es aconsejable normalizar los objetivos
  - Valores altos de hipervolumen son mejores que bajos



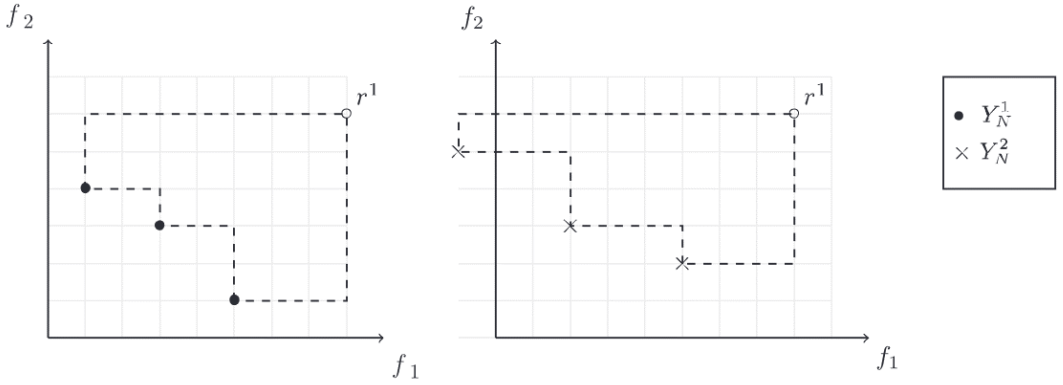


# Métricas para algoritmos multiobjetivo

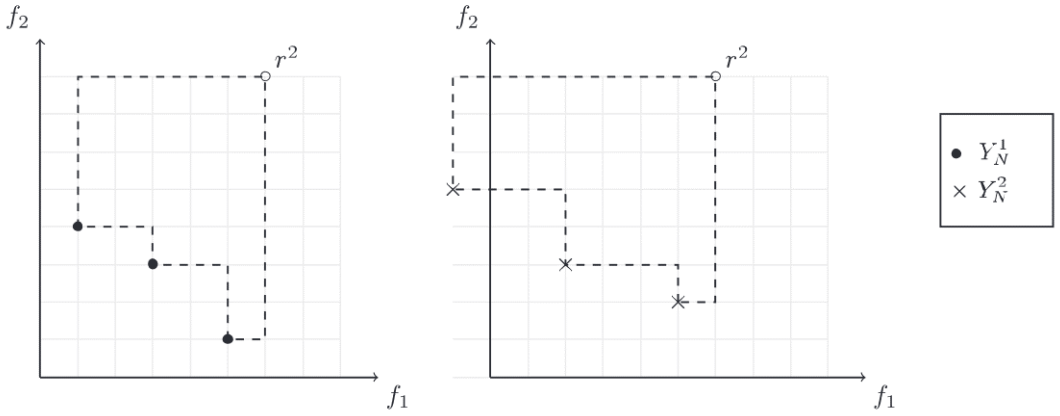
## Indicadores unarios: hipervolumen

- Uno de los problemas que puede presentar el hipervolumen es que el resultado depende del punto de referencia

- Utilizando el punto de referencia $r^1$=(8,6), el hipervolumen de la aproximación $Y^1$ es 25, mientras que el de $Y^2$ es 24
  - ¿$Y^1$ es mejor que $Y^2$?




- Utilizando el punto de referencia $r^2$=(6,8), el hipervolumen de la aproximación $Y^1$ es 25, mientras que el de $Y^2$ es 30
  - ¿$Y^2$ es mejor que $Y^1$?



# Contenidos

1. Metaheurísticas: introducción y clasificación
2. Algoritmos genéticos
3. Introducción a la búsqueda local
4. Algoritmos meméticos
5. Introducción a la optimización multiobjetivo
6. **Conclusiones**

# Conclusiones

- Hemos descrito qué son las metaheurísticas y cuándo es aconsejable su utilización
- Hemos visto qué es un Algoritmo Genético y cómo se puede utilizar para resolver problemas complejos, explicando operadores genéticos para diferentes esquemas de codificación, y profundizando en la importancia del adecuado equilibrio entre diversidad y convergencia
- Los AGs por sí solos no siempre consiguen buenos resultados, pero son flexibles y pueden ser mejorados con conocimiento sobre el problema y combinados con otras técnicas, como por ejemplo población inicial heurística o búsqueda local
- Hemos definido la búsqueda local y los operadores de vecindad, además de los dos métodos básicos (escalada de máximo gradiente y escalada simple)
- Hemos visto cómo se puede combinar un algoritmo genético con búsqueda local, lo cual se denomina Algoritmo Memético (o algoritmo genético híbrido)
- Hemos formulado el problema general de la optimización multiobjetivo, y hemos revisado el NSGA-II, que es la metaheurística más conocida. Por último, hemos descrito el hipervolumen, que es el indicador más utilizado para evaluar el rendimiento de los algoritmos multiobjetivo

# Un buen libro sobre metaheurísticas: *[Talbi, 2009]*