Tema 4. Otros tipos de búsqueda. Algoritmos Evolutivos

Objetivos

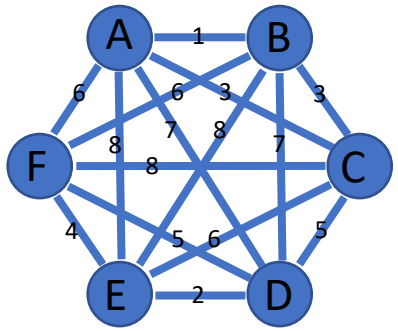
1. Conocer otros paradigmas de búsqueda, en concreto un tipo particular de Metaheurística: los Algoritmos Genéticos
2. **Demostrar como un Algoritmo Genético puede resolver de forma aproximada problemas de optimización complejos**

Contenidos

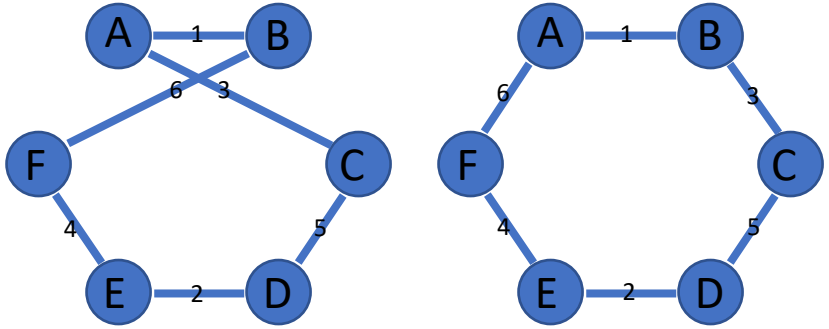
1. Algunos ejemplos motivadores
2. ¿Qué es un Algoritmo Genético?
3. El Algoritmo Genético Simple
4. Algoritmos Genéticos con representaciones no binarias
5. Ejemplos de aplicación de AGs

4.1 Un ejemplo: el 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

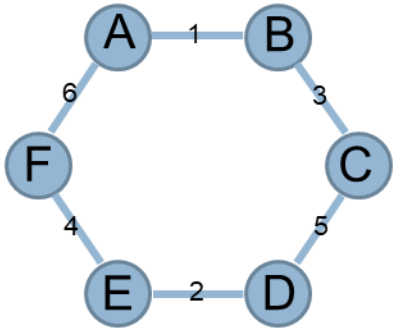


Dos soluciones óptimas (Coste = 21)

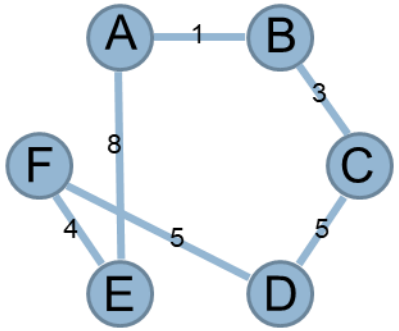
Métodos de resolución el TSP

- Enumeración exhaustiva: en el TSP es factible enumerar las soluciones y quedarse con la mejor
  - N = 6 5! = 120 soluciones candidatas
  - N = 61 60! > 10^{80} ≅ número de partículas elementales en el universo
- Búsqueda en espacios de estados
  - El árbol de búsqueda crece de forma exponencial, aun con el mejor heurístico
- Otra opción: generar un conjunto de soluciones y someterlas a un proceso evolutivo, combinando unas con otras y sometiéndolas a modificaciones. Estas son las claves de los Algoritmos Genéticos
  - Es necesario un sistema de codificación y decodificación de soluciones
  - Y buenos algoritmos de combinación y modificación
  - Y buenas estrategias de selección
  - Y alguna cosa más . . . .

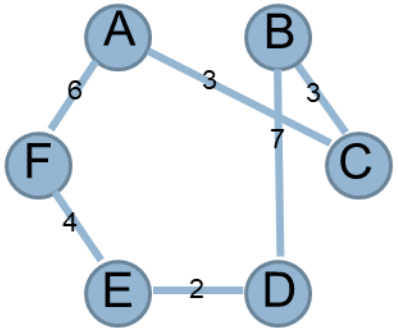
Codificación con permutaciones para el TSP



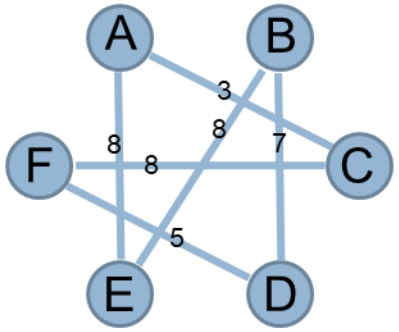
A B C D E F



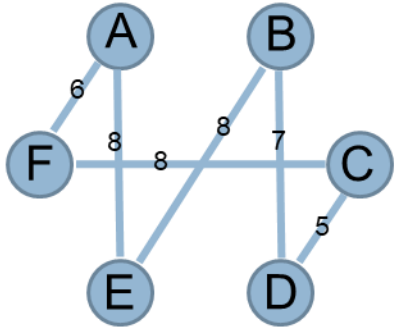
A B C D F E



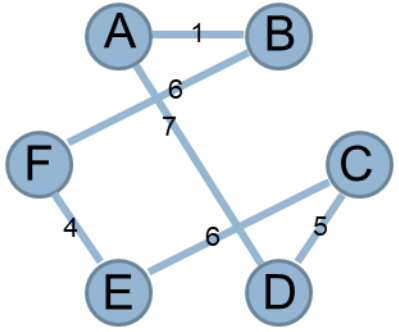
A C B D E F



A C F D B E



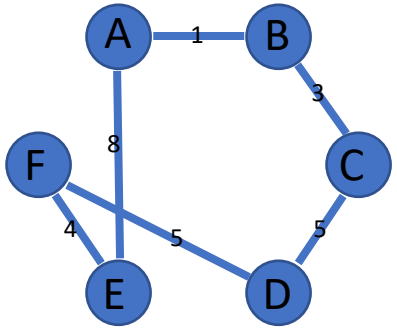
A E B D C F



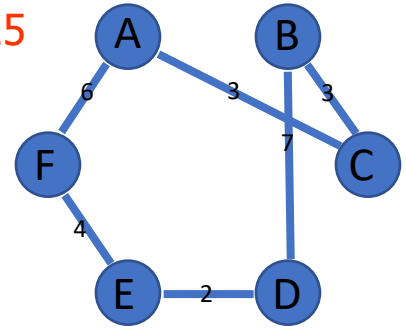
A B F E C D

Combinación de soluciones

Ejemplo de combinación no productiva

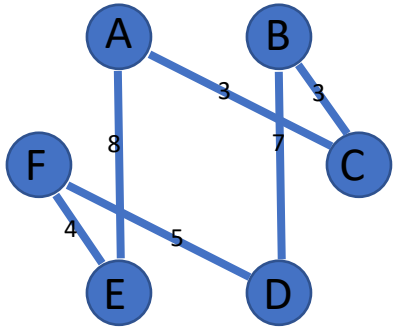


Padre_1: 26



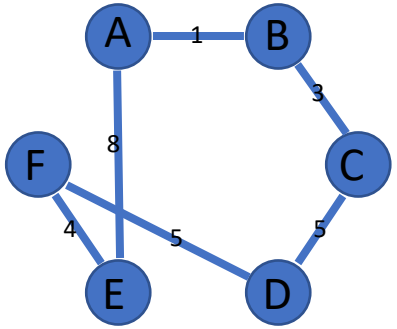
Padre_2: 25

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



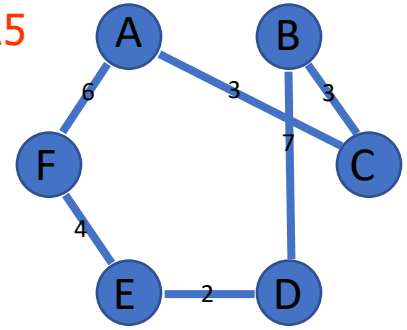
Hijo: 29

Combinación de soluciones
Ejemplo de combinación productiva

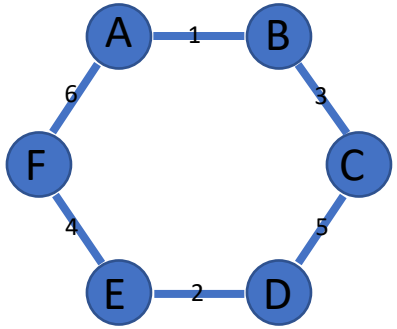


Padre_1: 26

Padre_2: 25



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



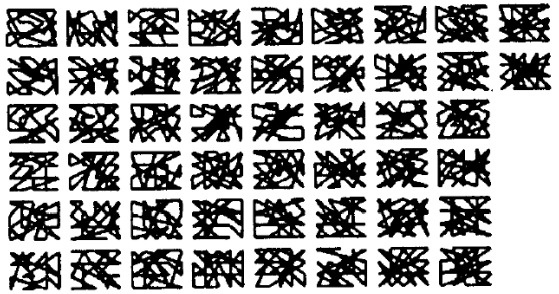
Hijo: 21

Propiedades de la codificación y de los operadores

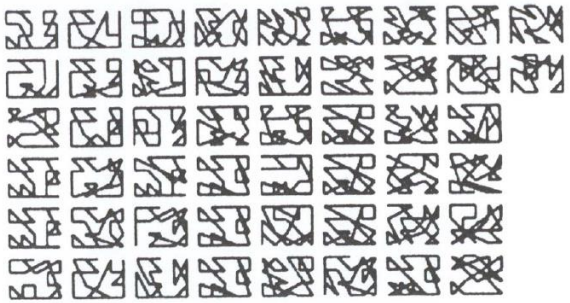
- Propiedades de los métodos de codificación
  - Deben ser exhaustivos
  - Deben permitir operadores eficientes
- Propiedades de los operadores
  - Que produzcan soluciones factibles
  - Que no sean muy costosos computacionalmente
  - Que los hijos hereden características relevantes de los padres
  - Y alguna más . . . .

Evolución de una población de soluciones

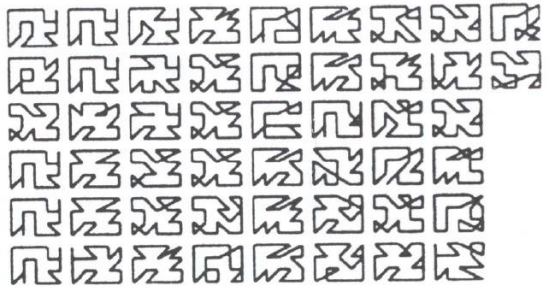
Ejemplo para una instancia del TSP



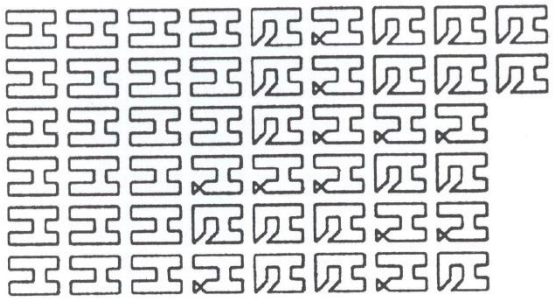
(0)



(10)



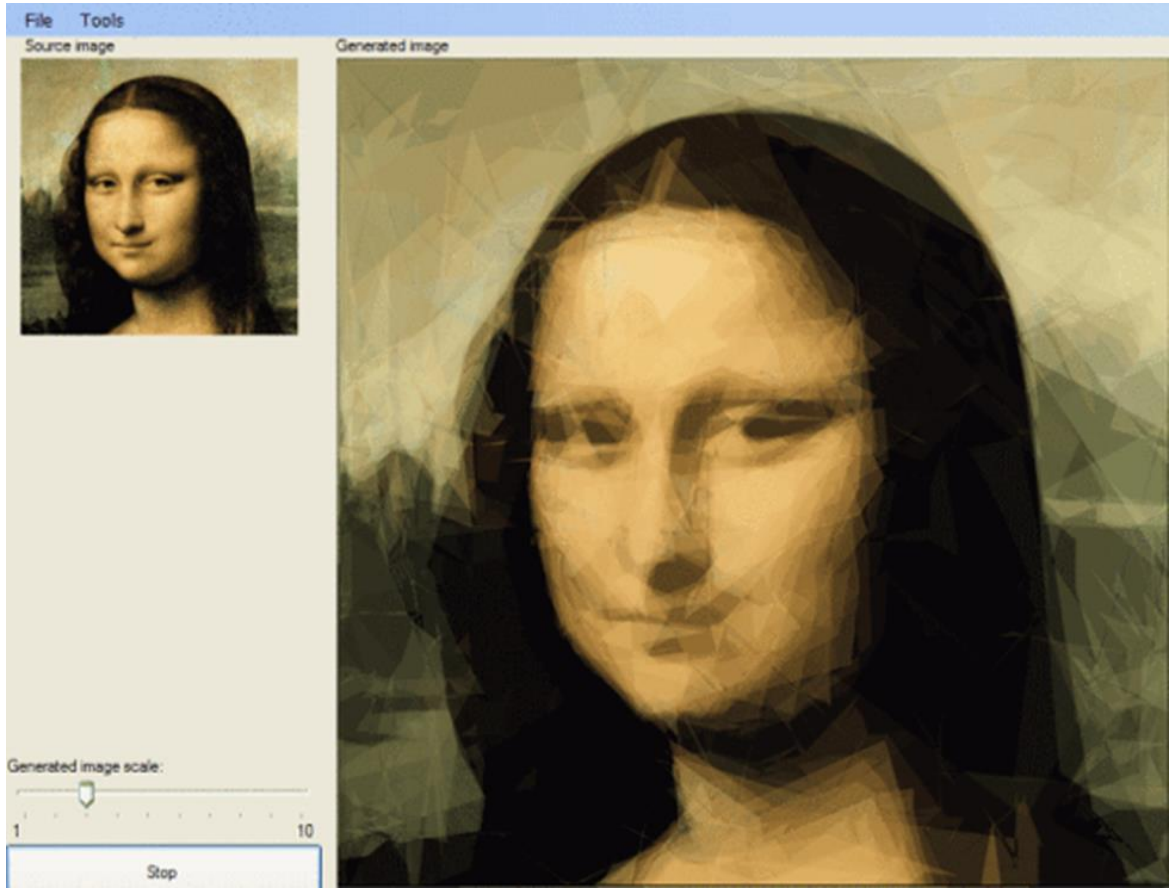
(30)



(70)

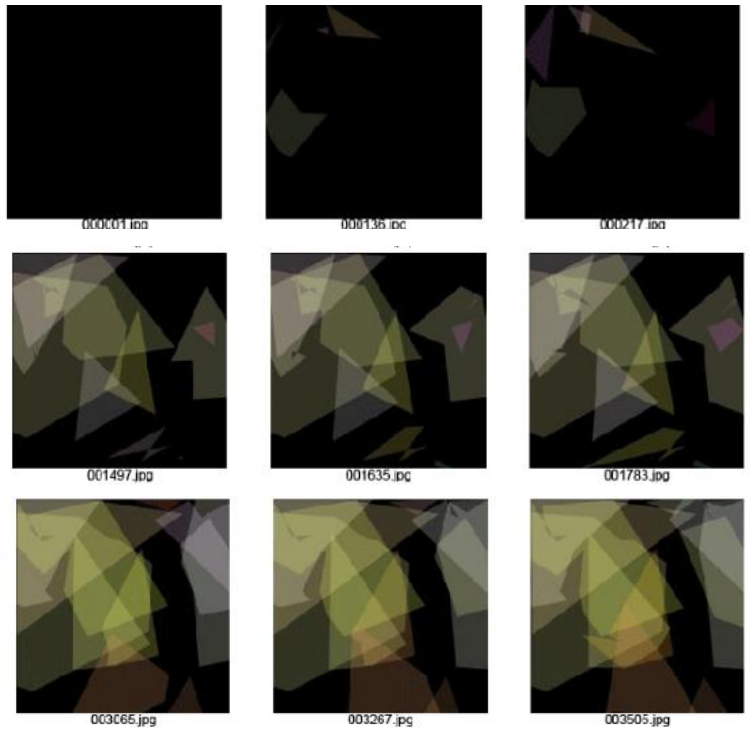
Otro ejemplo: *¿Se podría pintar una réplica de Mona Lisa usando sólo polígonos semitransparentes?*

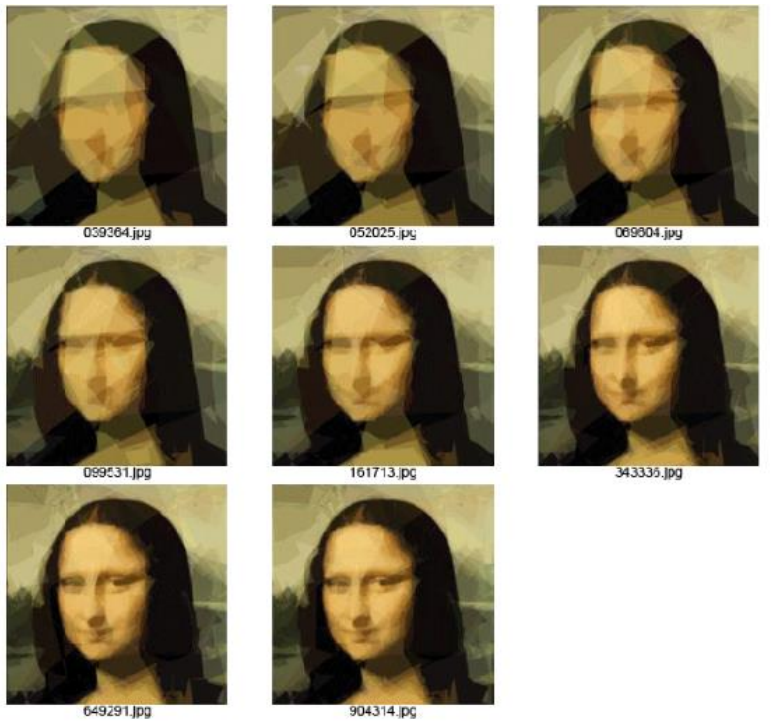
http://rogeralsing.com/2008/12/07/genetic-programming-evolution-of-mona-lisa/
https://github.com/peterbraden/genetic-lisa



Mona Lisa evolutiva

http://rogeralsing.com/2008/12/07/genetic-programming-evolution-of-mona-lisa/
https://github.com/peterbraden/genetic-lisa

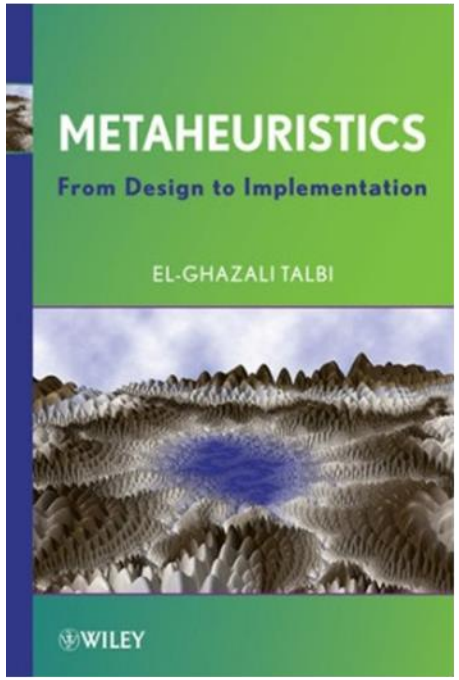


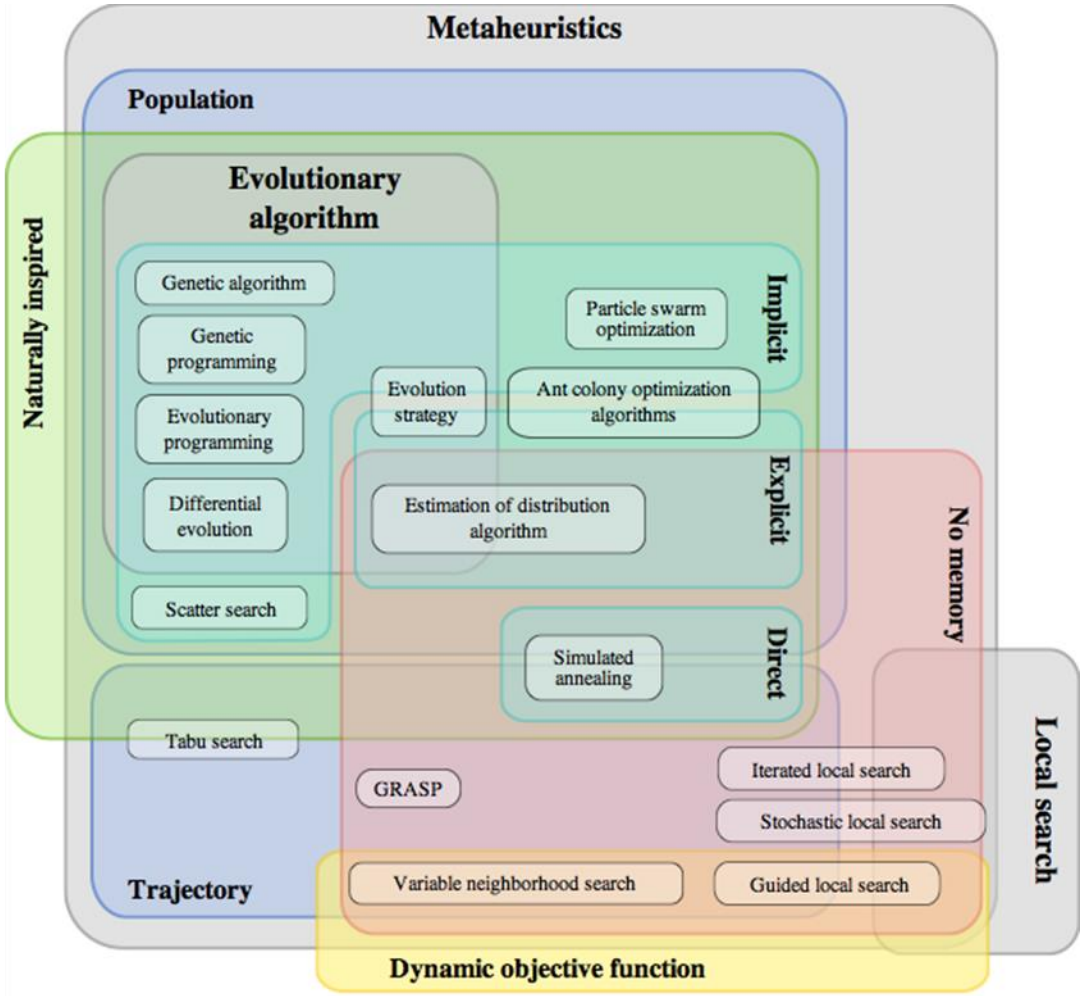


4.2. ¿Qué es un Algoritmo Genético?

- Los Algoritmos Genéticos son algoritmos de optimización y búsqueda que se basan en los mecanismos de la evolución natural, en particular en la selección natural y la herencia genética.
- Fueron introducidos por John Holland y sus colaboradores en la Universidad de Michigan a principios de los años 70 [*Holland 1975. Adaptation in Natural and Artificial Systems*].
- Se clasifican dentro de las denominadas metaheurísticas.

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





La metáfora evolutiva

El lenguaje que se utiliza en Algoritmos Genéticos

- **Evolución Natural**
  - Ecosistema o Entorno
  - **Cromosoma** o Genotipo
  - **Individuo** o Fenotipo
  - Adaptación al Entorno
  - Supervivencia, Reproducción, **Cruce**, **Mutación**

- **Evolución Artificial**
  - **Instancia del problema**
  - Cadena de símbolos
  - **Solución potencial**
  - Calidad de la solución, o **Fitness**
  - **Selección**, **Combinación, Cruce, Mutación, Reemplazamiento**

Componentes principales de un Algoritmo Genético

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

Un Algoritmo Genético convencional

```
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
```

Un Algoritmo Genético convencional

Versión Rusell&Norvig, 2022

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights ← WEIGHTED-BY(population, fitness)
    population2 ← empty list
    for i = 1 to SIZE(population) do
      parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child ← REPRODUCE(parent1, parent2)
      if (small random probability) then child ← MUTATE(child)
      add child to population2
    population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

Figure 4.7 A genetic algorithm. Within the function, *population* is an ordered list of individuals, *weights* is a list of corresponding fitness values for each individual, and *fitness* is a function to compute these values.

El esquema de codificación

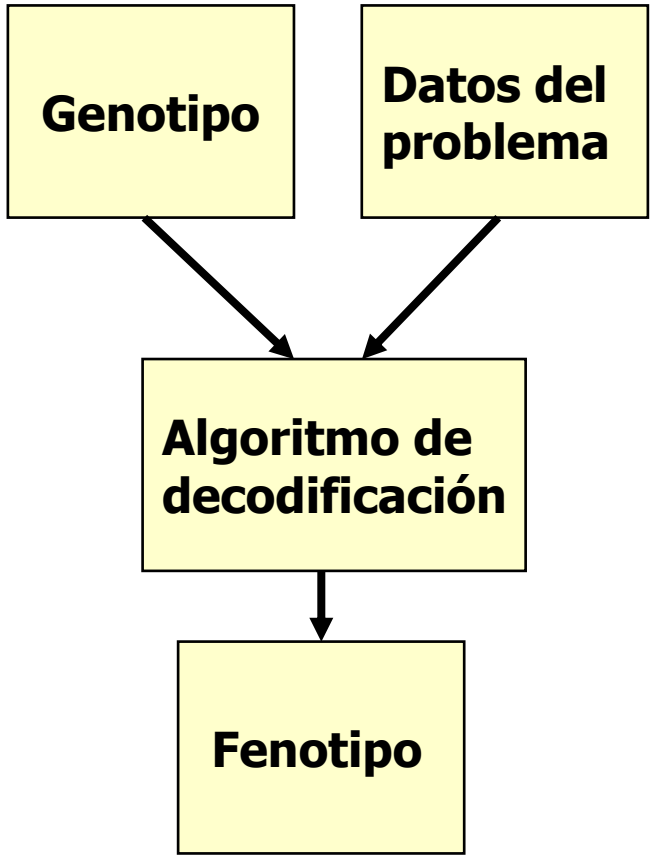
- Codifica una solución potencial, y el algoritmo de evaluación la debe construir de forma eficiente (en tiempo polinomial)
- Debe tener buena capacidad de representación
  - Al menos de un subconjunto de soluciones buenas
  - A ser posible con una correspondencia de uno a uno
- Debe ser fácil diseñar operadores genéticos que
  - Generen cromosomas factibles
  - Trasladen características relevantes de padres a hijos
  - No sean demasiado costosos computacionalmente
  - Los hijos tengan una “alta” probabilidad de mejorar
- La codificación más habitual es una cadena de símbolos: binaria, permutaciones, vectores de números, …

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
    - Una permutación de las ciudades: (5 3 4 1 2)
  - Job Shop Scheduling
    - Una permutación de las tareas: (1 3 2 4 5)
- Otras estructuras más complejas
  - Matrices
  - Árboles
    - Programación Genética
  - . . .

El algortimo 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 fácilmente del fenotipo

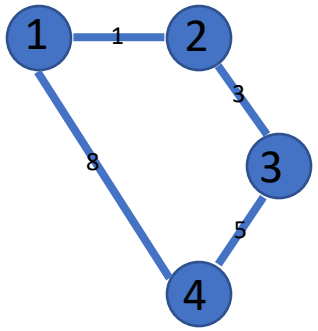


El algortimo de decodificación

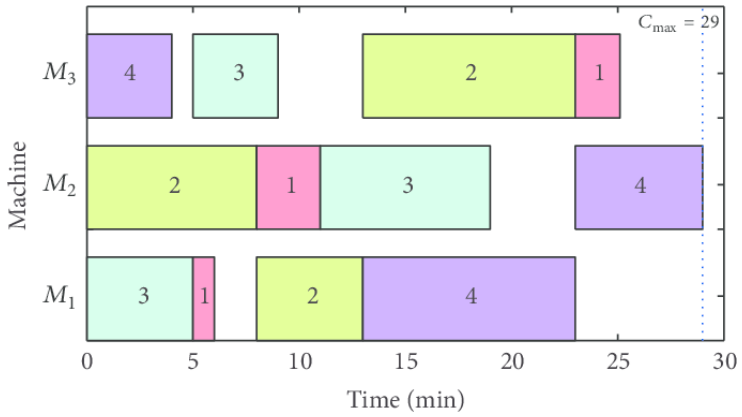
Ejemplos con permutaciones

(1 4 3 2)

Decodificador TSP



Decodificador JSP



4.3. El Algoritmo Genético Simple (SGA)

- Vamos a detallar el Algoritmo Genético más simple que se puede diseñar: el SGA
  - Codificación binaria (cadenas de bits)
  - Población inicial aleatoria
  - Cruce en un punto
  - Mutación simple
  - Selección proporcional al Fitness (ruleta)
  - Reemplazo generacional con aceptación incondicional (los hijos reemplazan a sus padres)

El SGA tiene interés especial porque está estudiado formalmente y tiene propiedades que ayudan a entender y a diseñar otras versiones más sofisticadas

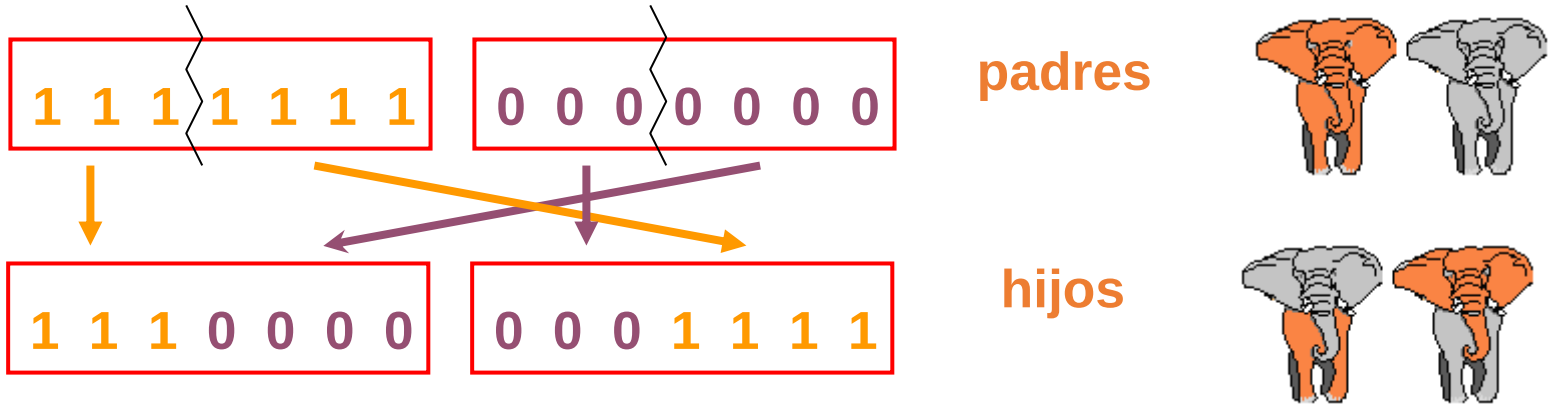
Codificación binaria

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

- 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

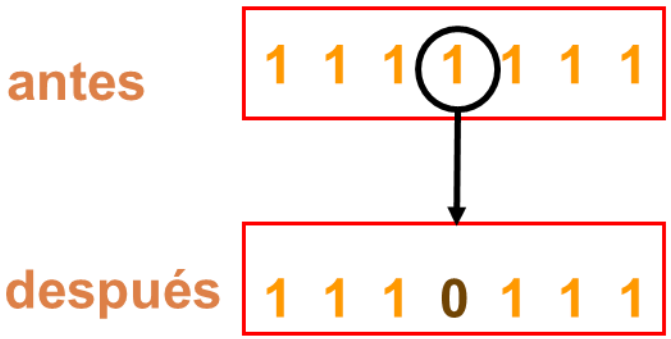
Cruce en un punto

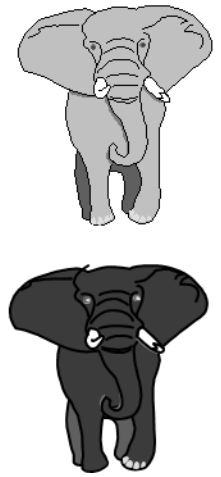
- Genera dos hijos de dos padres (se suele aplicar con una alta probabilidad Pc: 0.6 a 1)
- Cada hijo hereda características de sus padres
- Es el componente de EXPLOTACIÓN y es posiblemente el operador más importante de un Algoritmo Genético



Mutación simple

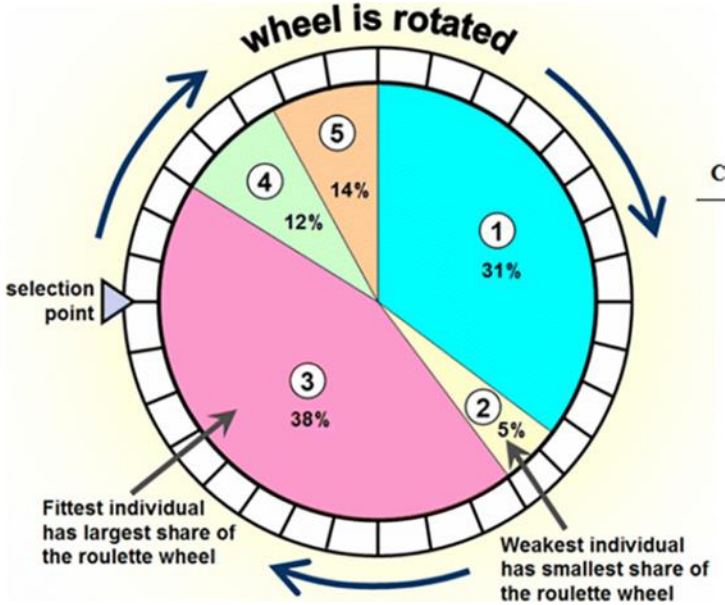
- Modifica un único bit (gen) según la probabilidad de mutación (muy pequeña, alrededor de 0.01)
- Introduce características aleatorias en la estructura del cromosoma. De esta forma es posible introducir nuevo material genético en la población
- Es el componente de EXPLORACIÓN





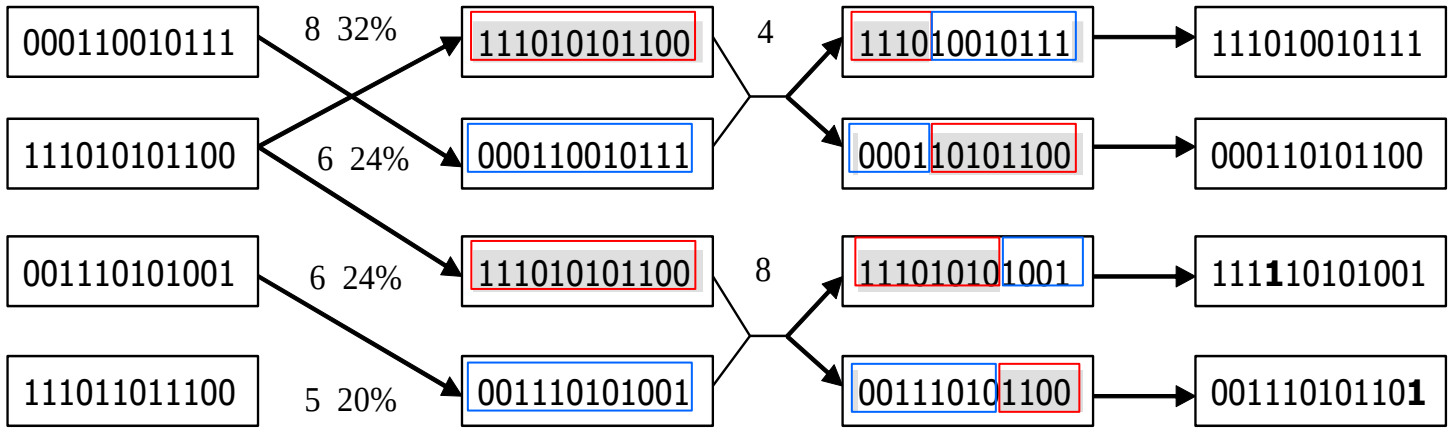
Selección proporcional al fitness (regla de la ruleta)

- 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()*



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

Un ejemplo de selección, cruce y mutación



a) P(t)

b) Fitness y probabilidad de ser elegido

c) P'(t) Tras la fase de selección

d) Punto de cruce

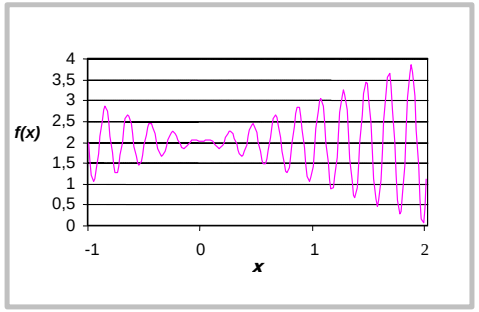
e) Tras aplicar el operador de cruce

f) P''(t) Tras aplicar el operador de mutación

Ejemplo de aplicación
Optimización numérica

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

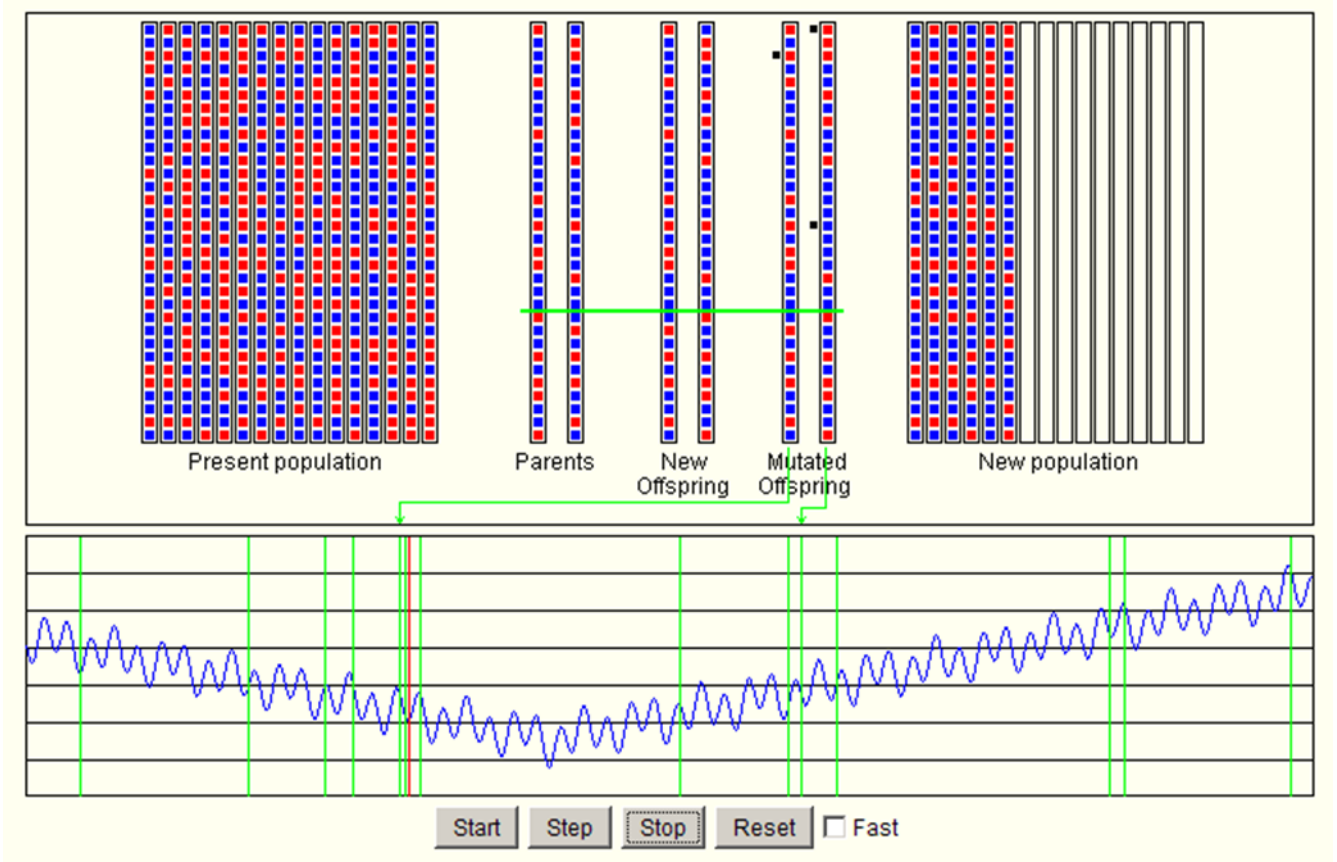
$f(x) = x*\sin(10\pi x)+2.0$ en $[-1,2]$



- 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 y determina el número de posibles cromosomas diferentes
    - Si la precisión es 10^6 la longitud debe ser 32, ya que $2^{31} \leq 10^6 \leq 2^{32}$

Visualizando el funcionamiento de un AG

http://www.obitko.com/tutorials/genetic-algorithms/



¿Por qué un algoritmo como SGA que maneja símbolos de manera ciega puede hacer algo útil?

- La respuesta está en la **Teoría de los Esquemas**
  - Un esquema es una cadena de símbolos del alfabeto {0,1,*}
    - Ejemplos: H_1=**101*00*, H_2=10*******1
  - Un esquema representa un conjunto de cromosomas
    - 01**101**10**00**1 y 10**101**00**00**0 pertenecen a H_1
  - La calidad de un esquema es la calidad media de los cromosomas que pertenecen a ese esquema
  - La longitud de un esquema H, δ(H), es la distancia desde el primer símbolo fijo hasta el ultimo símbolo fijo
    - δ(**101*00*) = 8-3 = 5
  - El orden de un esquema, O(H), es el número de símbolos fijos
    - O(**101*00*) = 5

Teorema fundamental de los AGs

- **Teorema de los Esquemas**. Los esquemas de orden bajo, distancia corta y calidad alta aumentan exponencialmente su representación en la población a medida que avanzan las generaciones.

- **Hipótesis de los Building Blocks**. Un AG obtiene una alta eficiencia gracias a la yuxtaposición de esquemas de orden bajo, distancia baja y calidad alta, denominados “building blocks”.

4.4. Diseño de Algoritmos Genéticos

- Algunas consideraciones importantes
  - Población inicial
  - Condición de parada
  - Ajuste de parámetros
  - Convergencia
  - Diversidad del material genético
  - Presión selectiva
  - Disrupción
  - Elitismo
  - . . . .

Convergencia de un AG

- Un AG se espera que tenga una convergencia hacia soluciones cada vez mejores a lo largo de las generaciones

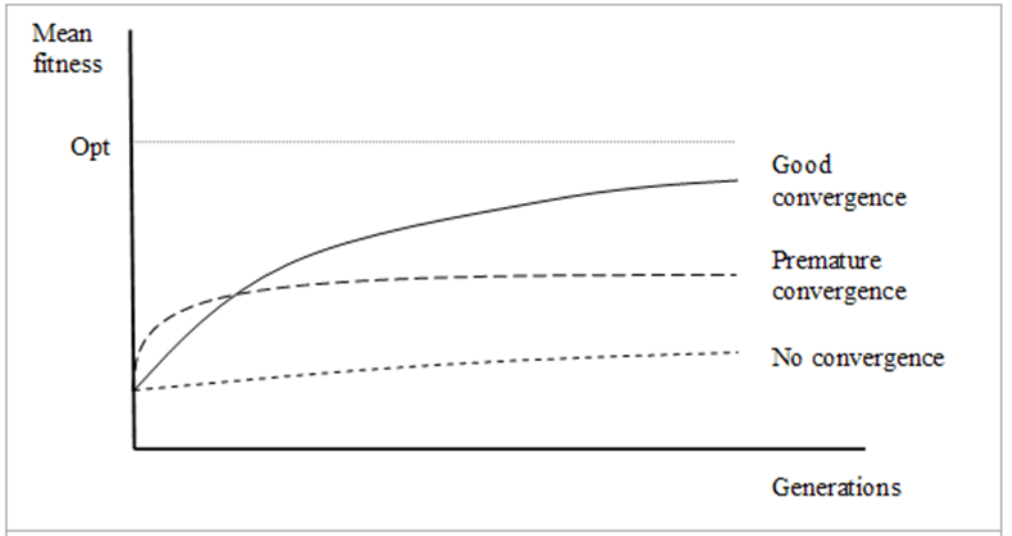


Figure 3. Three typical convergence patterns of the mean fitness of the population in a genetic algorithm.

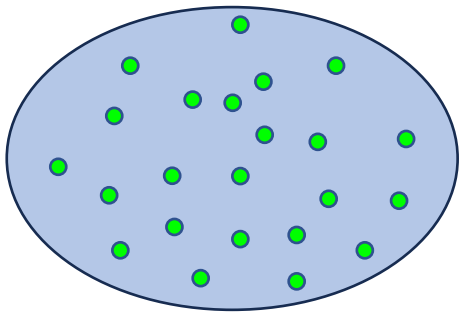
- Las claves son la diversidad, la presión selectiva y la disrupción

Una de la claves

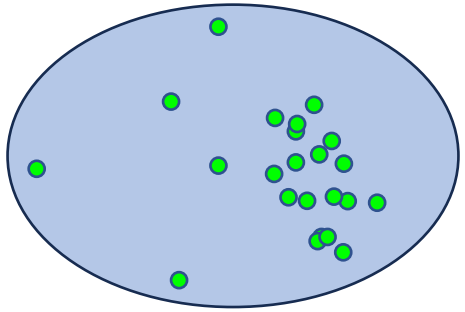
Equilibrio entre exploración y explotación

- EXPLORACIÓN: muestrear regiones desconocidas.
  - Exceso: búsqueda aleatoria, sin convergencia. Alta diversidad.
- EXPLOTACIÓN: muestrear de forma más intensa en la proximidad de las soluciones buenas.
  - Exceso: Tendencia a caer en máximos locales y convergencia prematura

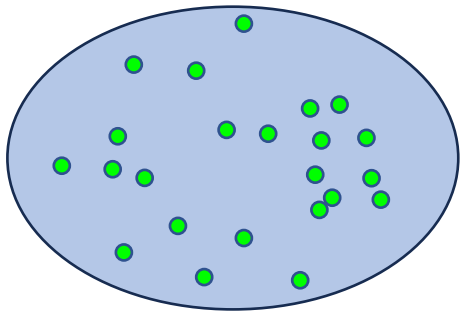
Búsqueda aleatoria



Convergencia prematura



Convergencia normal



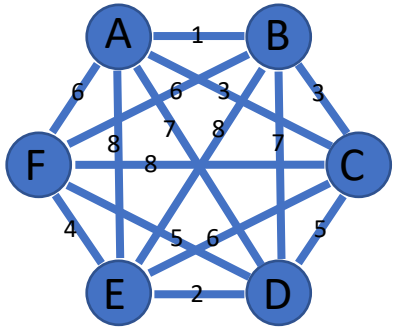
4.5. Aplicaciones de Algoritmos Genéticos

- Optimización combinatoria
  - Problema del viajante de comercio (Travelling 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

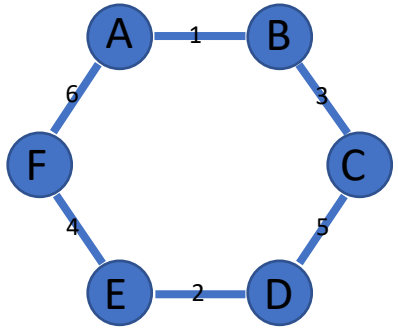
- Optimización numérica
  - Funciones max-min multimodales
  - Ajuste de parámetros
  - Generación y reconocimiento de imágenes
  - Entrenamiento de redes neuronales
  - Cálculo de estructuras
  - . . . . . .

Un ejemplo de aplicación
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 inicio, con el mínimo coste



Una instancia del TSP



Una solución óptima

AG para el TSP
Esquema de codificación

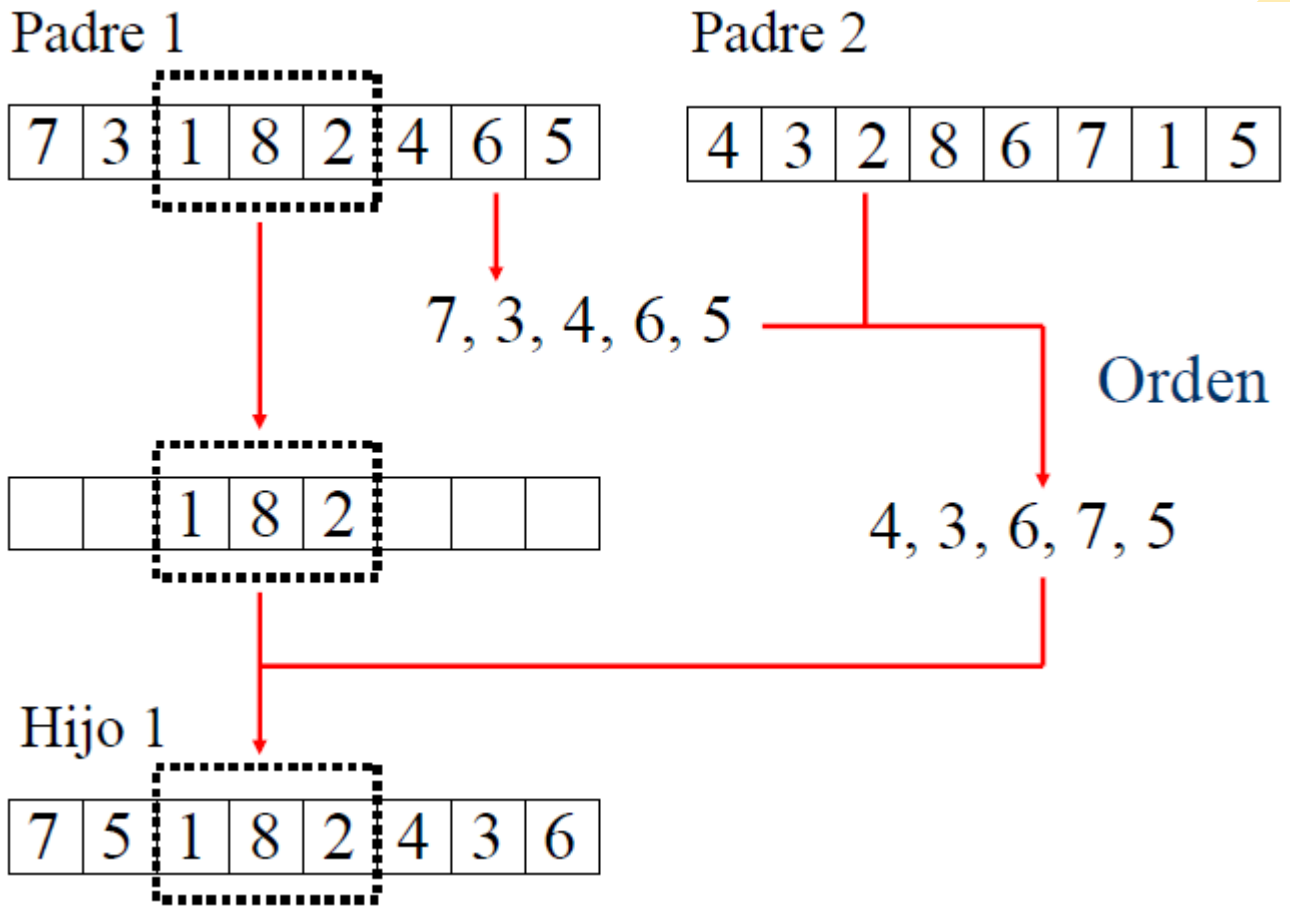
- Dos opciones
  - Codificación binaria
    - Un cromosoma es una cadena de n(log n) bits
    - Problema: factibilidad tras la inicialización, cruce y mutación. Requiere algún mecanismo de reparación
  - Permutaciones de las n-1 ciudades 1, … , n-1 (o de las n ciudades)
    - Un cromosoma es una permutación $(i_1,\ldots,i_{n-1})$ de las ciudades {1,2,…,n-1}. Requiere operadores de cruce más sofisticados
    - La evaluación es bastante simple: $(i_1,\ldots,i_{n-1})$ representa el recorrido $0 \rightarrow i_1$, $i_1 \rightarrow i_2$, $i_2 \rightarrow i_3$, . . ., $i_{n-1} \rightarrow 0$.

AG para el TSP con permutaciones
Operador de Cruce

- Consideraciones
  - Factibilidad: el resultado debe ser una permutación válida
  - Componente de EXPLOTACIÓN: los hijos deben heredar características relevantes de sus padres
    - ¿Cuáles son realmente esas características?, es decir ¿cómo son los esquemas?
- Algunos operadores de cruce independientes del dominio para codificaciones con permutaciones
  - Partial Mapping Crossover (PMX)
  - **Order Crossover (OX)**
  - Cycle Crossover (CX)
  - . . . .

AG para el TSP con permutaciones
Operador de Cruce. Un ejemplo (OX)

- Order Crossover (OX)
  - Copia una subsecuencia de símbolos del primer padre al hijo manteniendo el orden y la posición
  - El resto de símbolos ocupan las posiciones restantes manteniendo el orden que tienen en el segundo padre
  - Cada hijo hereda el orden y la posición de algunos genes de un padre y el orden relativo de los restantes genes del otro padre



AG para el TSP con permutaciones
Mutación, Inicialización y Evaluación

- Mutación
  - Intercambiar dos ciudades
  - Reordenar una subsecuencia pequeña, …
- Inicialización
  - Las permutaciones aleatorias en general no son buenas porque los mejores recorridos incluyen muchos de los caminos más cortos
  - En este caso los recorridos heurísticos, por ejemplo utilizando el heurístico del vecino más cercano, o Nearest Neighbour (NN) son mucho mejores
- Evaluación

  $T = (i_1,\ldots,i_{n-1})$

  $Coste(T) = coste(0,i_1) + coste(i_1,i_2) + \ldots + coste(i_{n-1},0)$

  $Fitness(T) = 1 / Coste(T)$

AG para el problema de las *N*-reinas
Codificación de los cromosomas

- Problema que consiste en colocar N reinas en un tablero de ajedrez de NxN casillas de tal forma que no se ataquen entre sí.

- **Codificación binaria:** secuencia de NxN ceros y unos, en donde un 1 indica la presencia de una reina en esa casilla.
  - Inconveniente: se pueden generar soluciones con un número de reinas distinto de N y con varias reinas en la misma fila o misma columna.
  - Inconveniente: Para N=8 hay 2^{64} cromosomas distintos.
  - Inconveniente: ¿Cómo diseñamos operadores de cruce y mutación eficientes?

0	**1**	0	0	0	0	0	0
0	0	0	**1**	0	0	0	0
0	0	0	0	0	0	**1**	0
1	0	0	0	0	0	0	0
0	0	0	0	0	**1**	0	0
0	0	**1**	0	0	0	0	0
0	0	0	0	**1**	0	0	0
0	0	0	0	0	0	0	**1**

AG para el problema de las *N*-reinas
Codificación de los cromosomas

- Otra opción: permutaciones de 1..N ([1] .. [N])
  - [i] representa la fila de la reina en la columna i

2	4	7	1	6	3	5	8

  - Ventaja: ninguna de las soluciones generadas tendrá varias reinas en la misma fila o misma columna, y todas tienen N reinas
  - Ventaja: Para N=8, hay 8! = 40320 posibles permutaciones y es seguro que una de ellas es una solución óptima
  - Ventaja: podemos utilizar operadores de cruce y mutación ya conocidos para otros problemas basados en permutaciones

AG para el problema de las *N*-reinas
Función de fitness

- Ante dos soluciones candidatas, ¿cómo decidir cuál es mejor?
  - Una opción: contar el número de pares de reinas que no se atacan entre sí (la solución óptima tendría fitness_1=$\sum_{i=1}^{N-1} i$)
  - Otra opción: contar el número de reinas que no son atacadas por ninguna otra (la solución óptima tendría fitness_2=*N*)

  - La primera opción es capaz de discriminar mejor entre las soluciones, y por eso da mejores resultados

AG para el problema de las *N*-reinas
Operadores genéticos

- Operador de cruce
  - Un operador genérico para permutaciones, como por ejemplo el cruce OX,
  - Un operador diseñado específicamente para el problema
- Operador de mutación
  - Uno genérico como por ejemplo intercambiar dos posiciones del cromosoma
  - Uno específico para el problema

Resumen

- Hemos visto qué es un AG y cómo se puede utilizar para resolver problemas numéricos o de optimización combinatoria complejos
- Hemos justificado por qué el AG binario es capaz de obtener resultados
- Hemos visto algunos applets y software que pueden ayudar a entender y diseñar AGs
- Debemos saber que los AGs son métodos débiles, y que deberían ser combinados con otras técnicas (por ejemplo búsqueda local) para mejorar su rendimiento