TEMA 1. INTRODUCCIÓN

1. TIPOS DE PROBLEMAS DE BÚSQUEDA

a. Obtener el camino más corto entre 2 puntos.
b. Obtener el plan de mínimo coste para repartir mercancías.
c. Asignar de manera óptima tareas a los trabajadores

2. CARACTERÍSTICAS DE LOS PROBLEMAS DE BÚSQUEDA

a. Son problemas NP-Hard.
b. Son algoritmos exactos ineficientes o imposibles de aplicar.

3. EJEMPLOS DE PROBLEMAS DE BÚSQUEDA

a. Problema de la mochila.
b. Viajante de comercio (TSP).
c. Optimizador de parámetros.
d. Controlador de Super Mario Bros.

4. CONCEPTO DE PROBLEMAS P

Es un conjunto de problemas donde podemos encontrar una solución en un tiempo razonable (polinomial).

5. CONCEPTO DE PROBLEMAS NP

Es un conjunto de problemas donde podemos comprobar en un tiempo razonable (polinomial) si una respuesta al problema es solución o no.

6. TODOS LOS PROBLEMAS EN P ESTÁN EN NP

Si encontramos una solución en tiempo razonable encontraremos en tiempo razonable si es respuesta correcta o no.

7. NO ESTÁ CLARO QUE UN PROBLEMA EN NP ESTÉ EN P

Comprobar una respuesta en un tiempo razonable no implica que dicha respuesta sea encontrada en un tiempo razonable.

8. **CONCEPTO DE FUNCIÓN OBJETIVO**
   Es una función que consiste en maximizar o minimizar una función.
9. **CONCEPTO DE OPTIMIZACIÓN DE FUNCIONES**
   Consiste en minimizar una función, es decir, obtener el menor valor de la función.
10. **TIPOS DE CODIFICACIÓN DE SOLUCIONES EN PROBLEMAS DE BÚSQUEDA**
    a. Binaria (100001): Consiste en obtener posibles soluciones a un problema utilizando los valores lógicos 1 y 0. Un ejemplo de problema en el que se utilice esta codificación de soluciones es en el problema de la mochila con elección simple.
    b. Entera (1 4 3 2): Consiste en codificar una posible solución a un problema utilizando valores enteros. Un ejemplo de problema en el que se utilice esta codificación de soluciones es en el problema de la mochila con elección múltiple.
    c. Ordinal (3 1 5 2 4 6): Consiste en codificar las posibles soluciones a un problema utilizando valores ordinales. Un ejemplo de problema en el que se utilice esta codificación de soluciones es en el problema del viajante de comercio.
    d. Real (2.45 1.01): Consiste en codificar las posibles soluciones a un problema utilizando valores reales. Un ejemplo de problema en el que se utiliza esta codificación de soluciones es en la optimización de funciones.
11. **CONCEPTO DE ALGORITMOS APROXIMADOS**
    Son algoritmos que aportan soluciones cercanas a la óptima en problemas complejos (problemas NP) en un tiempo razonable.

12. **CUANDO UTILIZAR ALGORITMOS APROXIMADOS**
    a. No existe un método exacto de resolución.
    b. Existe un método exacto pero requiere mucho tiempo de cálculo.
    c. No se necesita la solución óptima sino una solución lo suficientemente buena en un tiempo aceptable.
13. **TIPOS DE ALGORITMOS APROXIMADOS**
    a. Heurísticas: Son algoritmos aproximados que dependen del problema.
    b. Metaheurísticas: Son algoritmos aproximados más generales que las heurísticas y aplicables a gran variedad de problemas.
14. **CARACTERÍSTICAS DE LAS METAHEURÍSTICAS**
    a. Rapidez en la resolución de problemas más complejos.
    b. Robustez en los algoritmos.
    c. Metáforas naturales.
15. **PROPIEDADES DE LAS METAHEURÍSTICAS**
    a. Son una familia de algoritmos aproximados.
    b. Suelen ser procedimientos iterativos.
    c. Guían una heurística subordinada de búsqueda.
    d. Combinan diversificación (exploración global) con intensificación (exploración local).
16. **VENTAJAS DE LAS METAHEURÍSTICAS**
    a. Son algoritmos de propósito general.
    b. Tienen gran éxito.
    c. Son fáciles de implementar.
    d. Son fáciles de paralelizar.
17. **DESVENTAJAS DE LAS METAHEURÍSTICAS**
    a. Son aproximados, pero no exactos.
    b. Son estocásticos.
    c. No tienen una base teórica establecida.

18. **CARACTERÍSTICAS DE LA BÚSQUEDA DE ALGORITMOS APROXIMADOS**
    a. Se usa para construir o mejorar soluciones a un problema.
    b. Se usa para obtener la solución óptima o las soluciones casi-óptimas.
19. **ELEMENTOS DE LOS ALGORITMOS APROXIMADOS**
    a. Solución: Es una representación de la solución al problema.
    b. Entorno: Representa las soluciones cercanas a una posible solución al problema.
    c. Movimiento: Es la transformación de la solución actual en otra posible solución al problema.
    d. Evaluación: Es la factibilidad de la solución y la función objetivo.
20. **HEURÍSTICA DE GREEDY**
    a. Trata de construir una solución eligiendo de manera iterativa los elementos de menor coste.
    b. La solución encontrada no tiene por qué ser óptima.
    c. Ejemplo de problema: Viajante de comercio.

TEMA 2. MÉTODOS BASADOS EN TRAYECTORIAS

1. **CONCEPTO DE BÚSQUEDA BASADA EN TRAYECTORIA**
   Término local que refleja el concepto de proximidad entre las soluciones alternativas de un problema.
2. **CONCEPTO DE SOLUCIONES VECINAS EN BÚSQUEDA BASADA EN TRAYECTORIAS**
   Todas las soluciones incluidas en el entorno de la solución actual, delimitado por un operador de generación de soluciones.

3. **ALGORITMOS BASADOS EN TRAYECTORIAS**
   Efectúan un estudio local del espacio de búsqueda, analizando el entorno de la solución actual para decidir cómo continuar el recorrido de la búsqueda.
4. **CONCEPTO DE BÚSQUEDA LOCAL**
   Es un proceso que, dada una solución en un recorrido, selecciona iterativamente una solución de su entorno para continuar la búsqueda.
5. **INGREDIENTES DE UNA BÚSQUEDA LOCAL**
   a. Codificación y Evaluación.
   b. Selección de la solución inicial.
   c. Selección de las soluciones vecinas.
   d. Elección de la estrategia de exploración del vecindario.
   e. Elección del criterio de parada.
6. **MÉTODOS BÁSICOS DE BÚSQUEDA LOCAL**
   a. Escalada Simple: La clave es seleccionar cualquier operación que suponga una mejora.
   b. Escalada por la máxima pendiente: La clave es elegir la operación que suponga una mejora entre todas las soluciones vecinas.
7. **PROBLEMA DE LA BÚSQUEDA LOCAL**
   Caer en óptimos locales.
8. **SOLUCIONES AL PROBLEMA DE LA BÚSQUEDA LOCAL**
   a. Permitir movimientos que empeoren la solución actual: Enfriamiento Simulado, Búsqueda Tabú, etc.
   b. Modificar la estructura de entornos: Búsqueda Tabú, Búsqueda en entornos variables, etc.
   c. Volver a comenzar la búsqueda desde otra solución actual: Búsqueda local iterativa, etc.

9. EJEMPLOS DE METAHEURÍSTICAS BASADAS EN TRAYECTORIAS

a. Enfriamiento Simulado.
b. Búsqueda Tabú.
c. Búsqueda Local Iterativa.
d. Búsqueda por Entornos Variables (VNS).
e. GRASP.

10. ENFRIAMIENTO SIMULADO

a. Es un algoritmo de Escalada Estocástico: Consiste en la elección de un sucesor, que podría ser peor a la solución actual, de entre todos los posibles sucesores según una distribución de probabilidad.
b. Utiliza un algoritmo de búsqueda local (búsqueda por entornos) con un criterio probabilístico de aceptación de soluciones.
c. Utiliza una función que disminuye la probabilidad de moverse hacia peores soluciones, conforme avanza la búsqueda.
d. Evita que la búsqueda local finalice en óptimos locales, permitiendo movimientos hacia soluciones peores.
e. Se basa en diversificar al principio e intensificar al final.
f. Aplicación: Minimizar una función objetivo.
g. Existen distintos tipos de enfriamiento

$$\text{Exponencial} : T = \alpha^{it} T_0 \,/\, T = \alpha T$$
$$\text{Lineal} : T = T_0 - n \cdot it$$
$$\text{Boltzmann / Logarítmica} : T = T_0/(1+\log(1+it))$$
$$\text{Cauchy} : T = T_0 \,/\, (1+it)$$

h. No es conveniente usar un valor fijo que sea independiente al problema.

i. Para converger al óptimo global se requiere enfriamiento lento.

11. BÚSQUEDA TABÚ

a. Permite aceptar soluciones que empeoran la solución actual.
b. Es un algoritmo de búsqueda local basado en el uso de memoria.
c. La memoria se usa para no repetir la trayectoria de búsqueda.
d. Se vuelve más potente incluyendo memoria a largo plazo y estrategias de reinicialización asociadas.
e. Existen 2 tipos de memoria:
    i. Memoria a corto plazo: Se utiliza para evitar ciclos.
    ii. Memoria a largo plazo: Se utiliza para intensificar la búsqueda en buenas regiones y diversificar la búsqueda hacia nuevas regiones.

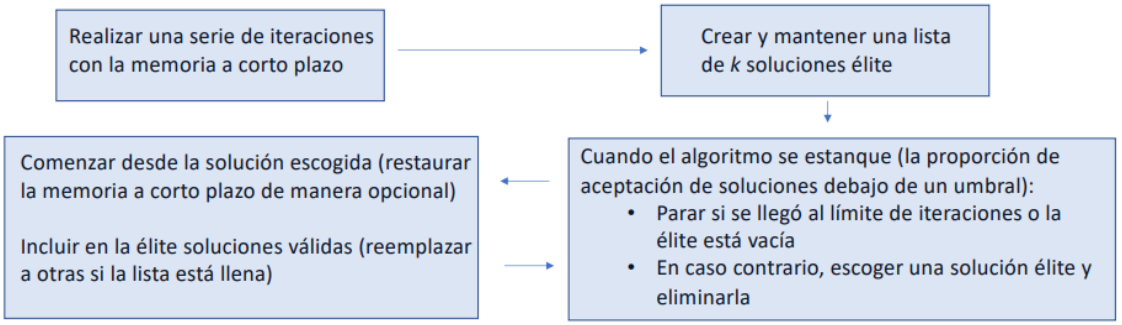
12. PRINCIPIO DE LA BÚSQUEDA TABÚ

a. Es mejor una mala decisión basada en información que una buena decisión al azar.
b. Una buena elección fruto del azar no proporcionará ninguna información para posteriores acciones.

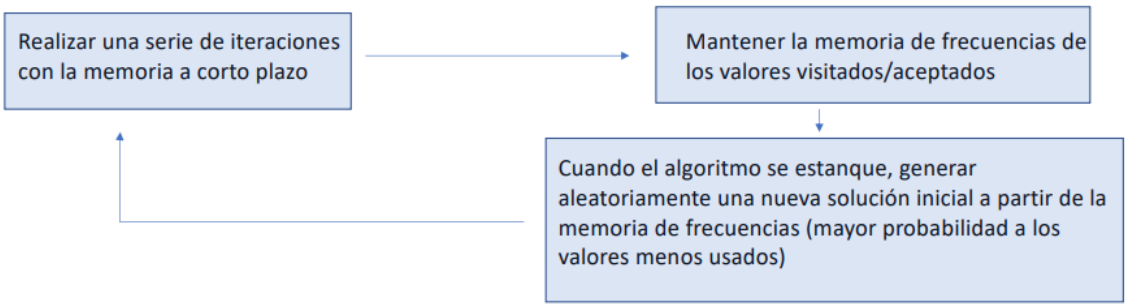
13. PROCEDIMIENTO ITERATIVO DE LA BÚSQUEDA TABÚ

1. Solución actual
2. Definir vecindario
3. Evaluar vecindario
4. Elegir el mejor No-Tabú. El movimiento siempre se acepta a pesar de que la nueva solución sea peor que la anterior

14. ENFOQUE SIMPLE DE INTENSIFICACIÓN EN LA BÚSQUEDA TABÚ



15. ENFOQUE SIMPLE DE DIVERSIFICACIÓN EN LA BÚSQUEDA TABÚ



16. BÚSQUEDA LOCAL ITERATIVA

a. Su base es la aplicación repetida de un algoritmo de búsqueda local a una solución actual que se obtiene por mutación de un óptimo local previamente encontrado.
b. Existen 4 componentes principales:
   i. Solución actual.
   ii. Procedimiento de búsqueda local.
   iii. Procedimiento de modificación.
   iv. Criterio de aceptación de a cuál solución se le aplica la modificación.

17. **BÚSQUEDA POR ENTORNOS VARIABLES**
    a. Se basa en la idea de un cambio sistemático de entorno o vecindad dentro de una búsqueda local (aumentando el tamaño cuando la búsqueda no avanza).
    b. Se basa en 3 hechos:
        i. Un mínimo local en un entorno no lo es necesariamente en otro.
        ii. Un mínimo global lo es en todos los entornos.
        iii. En muchos problemas, los mínimos locales con los mismos o distintos entornos están relativamente cerca.
18. **GRASP**
    a. Es un método multiarranque en el que cada iteración construye una solución greedy a la que aplicada una búsqueda local que toma dicha solución como punto de partida.
    b. Este proceso se repite varias veces y la mejor solución encontrada sobre todas las iteraciones GRASP se devuelve como la salida.
    c. Utiliza un procedimiento iterativo: Greedy + Búsqueda Local.

TEMA 3. ALGORITMOS EVOLUTIVOS

1. CONDICIONES PARA QUE OCURRAN LOS PROCESOS EVOLUTIVOS

a. Un individuo tiene la habilidad de reproducirse.
b. Existe una población de individuos que pueden reproducirse.

c. Existe alguna habilidad entre los individuos que se reproducen
d. Algunos individuos sobreviven en el entorno gracias a dicha habilidad.
e. El proceso no tiene memoria.

2. COMPUTACIÓN EVOLUTIVA

a. Es una simulación del proceso evolutivo en un ordenador.
b. Esta técnica de optimización probabilística mejora, con cierta frecuencia, a otros métodos clásicos en problemas difíciles.
c. Está compuesta por modelos de evolución basados en poblaciones cuyos individuos representan soluciones a problemas.
d. Se basa en principios darwinianos: Reproducción y Selección Natural.
e. Existen 4 paradigmas básicos:
    i. Algoritmos Genéticos: Utilizan operadores genéticos sobre cromosomas.
    ii. Estrategias de Evolución: Enfatizan los cambios de comportamiento al nivel de los individuos.
    iii. Programación Evolutiva: Enfatizan los cambios de comportamiento al nivel de las especies.
    iv. Programación Genética: Evoluciona expresiones que representan programas.

3. ALGORITMOS GENÉTICOS

a. Son algoritmos de optimización, búsqueda y aprendizaje inspirados en los procesos de evolución natural y evolución genética.
b. Utilizan los siguientes mecanismos:
    i. Sobreviven los organismos con mejor capacidad dentro de una población.

ii. Secuencias de caracteres para representar el ADN.
iii. Métodos aleatorios para generar la población y su reproducción.

c. Existen 4 condiciones necesarias para la evolución:
i. Entidad capaz de reproducirse.
ii. Población formada por dichas entidades.
iii. Variedad en la reproducción.
iv. Diferencias en la habilidad de las entidades que permiten sobrevivir en base al medio ambiente:
1. Individuos con mayor éxito tienen mayor probabilidad de reproducirse y generar un mayor número de descendientes.
2. Genes de individuos mejor adaptados se propagarán en sucesivas generaciones.

4. CONSTRUCCIÓN DE UN ALGORITMO GENÉTICO

a. Establecer el tipo de representación.
b. Decidir cómo inicializar la población.
c. Diseñar una correspondencia entre genotipo y fenotipo.
i. Genotipo: Dominio del algoritmo.
ii. Fenotipo: Manifestación del genotipo.
d. Definir una forma de evaluar los individuos.
e. Diseñar un operador de mutación adecuado.
f. Diseñar un operador de cruce adecuado.
g. Definir un operador de selección de padres.
h. Establecer un reemplazo de los individuos.
i. Definir una condición de parada.

5. TIPOS DE MODELOS DE ALGORITMOS GENÉTICOS

a. Modelo Generacional: En cada generación se crea una población completa con nuevos individuos, pudiendo llevar o no elitismo (se mantiene el mejor individuo).

b. Modelo Estacionario: En cada generación se escogen 2 o más padres de la población y se les aplica operadores genéticos, de modo que los nuevos individuos compiten por entrar en la población. (se genera una convergencia rápida cuando se reemplazan los peores cromosomas de la población).

6. PASOS EN LA CONSTRUCCIÓN DE UN ALGORITMO GENÉTICO

a. Representación: Debemos disponer de un mecanismo para codificar un individuo como un genotipo.
b. Inicialización: Debemos establecer claramente cómo vamos a inicializar la población.

- Distribución uniforme sobre el espacio de búsqueda
  - Binaria: 0 ó 1 con probabilidad 0.5
  - Real: uniforme sobre un intervalo dado

- Elegir una población a partir de los resultados de una heurística previa

c. Evaluación: Representación de la aptitud de cada uno de los individuos.
d. Estrategias de selección: Debemos garantizar que los mejores individuos tienen mayor probabilidad de reproducirse frente a los individuos menos buenos.
   i. Selección aleatoria: Se escogen padres de manera aleatoria de entre la población:
      1. Con reemplazamiento
      2. Sin reemplazamiento
   ii. Selección por torneo: Para cada padre a seleccionar:

- Se escoge aleatoriamente *k* individuos (con reemplazamiento)
- Se selecciona el mejor de ellos
- *k* es el tamaño del torneo. A mayor *k* mayor presión selectiva y viceversa
- Con *k*=1 es equivalente a una selección aleatoria

- **Torneos grandes**: hay una presión selectiva elevada y los peores individuos apenas tienen oportunidades de reproducirse
  - **Caso particular**: elitismo global, donde participan todos los individuos y la selección se vuelve totalmente determinista

- **Torneos pequeños:** la presión selectiva disminuye y los peores individuos tienen más oportunidades de ser seleccionados

iii. Selección por ruleta

- A cada individuo se le asigna una parte proporcional a su función de aptitud de una ruleta
- La suma de todos los porcentajes es igual a la unidad
- Los mejores individuos tendrán una proporción de ruleta mayor que los peores
- Dados N individuos, la probabilidad asociada a su selección es:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

- Generalmente la población está ordenada en base a la función de aptitud por lo que las proporciones más grandes están al inicio de la ruleta
- La selección se realiza con un aleatorio en el intervalo [0, 1] y devuelve en individuo en esa posición de la ruleta

iv. Selección basada en ranking: La población se ordena de acuerdo a su función de aptitud y las probabilidades de selección se asignan en base a su ranking.

- Puede evitar una convergencia prematura
- Puede ser computacionalmente costosa (ordenar la población)
- El individuo peor tiene la posición 1 y el mejor la posición N
- Se aplica una selección por ruleta normal, pero aplicada a los rankings

v. Emparejamiento variado inverso

- Se selecciona aleatoriamente un padre
- Para el otro padre, se seleccionan N padres y se escoge el más lejano al primero
- Está orientado a generar diversidad

e. Operadores de cruce: Los hijos deberán heredar algunas características de los padres y se deben producir cromosomas válidos.
El cruce es un método para compartir información entre cromosomas, de modo que explota la información disponible y siendo el único que puede combinar lo mejor de los padres.

i. Cruce en 1 punto

- Seleccionar un punto al azar en ambos padres
- Cortar los padres en este punto
- Crear hijos intercambiando las colas
- Probabilidad de cruce típica en el rango (0.6, 0.9)

ii. Cruce en n puntos

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

iii. Cruce uniforme

- Lanzar un aleatorio [0, 1] para cada gen de los padres
- En base a dicho aleatorio, tomar el gen de uno u otro padre
- El otro hijo será el inverso

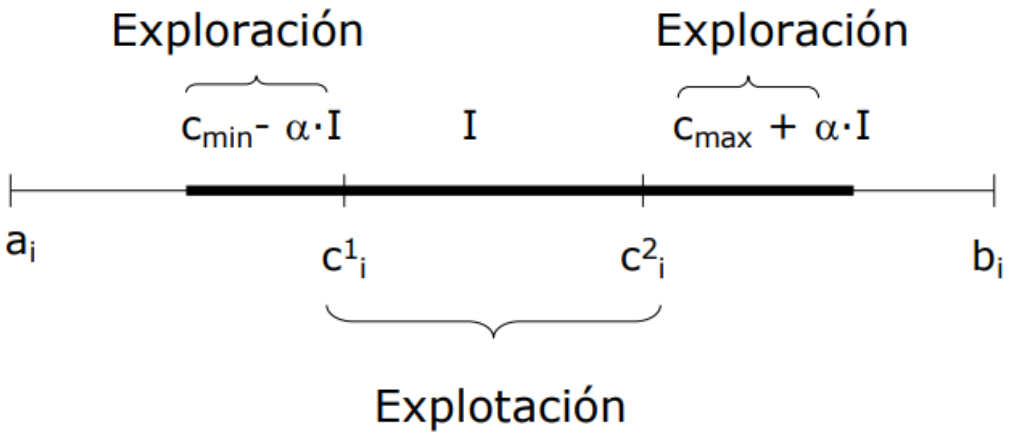
iv. Cruce aritmético simple

- El gen de la descendencia toma el valor medio de los genes de los padres
- Tiene la desventaja de que únicamente se genera un descendiente

v. Cruce BLX-ᵅ

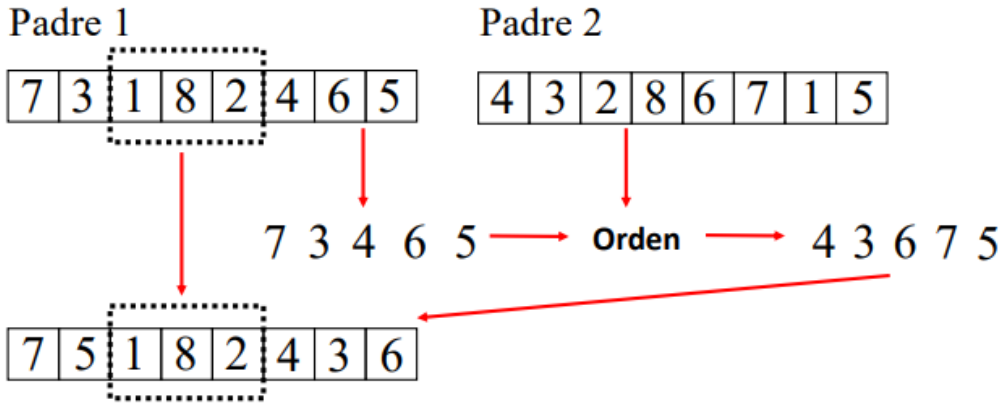
- Dados 2 cromosomas: $C_1 = (c_{11},\ldots, c_{1n})$ y $C_2 = (c_{21},\ldots, c_{2n})$
- BLX-α genera dos descendientes: $H_k = (h_{k1},\ldots, h_{ki},\ldots, h_{kn})$, k =1,2
- h_{ki} se genera aleatoriamente en el intervalo: $[C_{min} - I\cdot \alpha, C_{max} + I\cdot \alpha]$

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

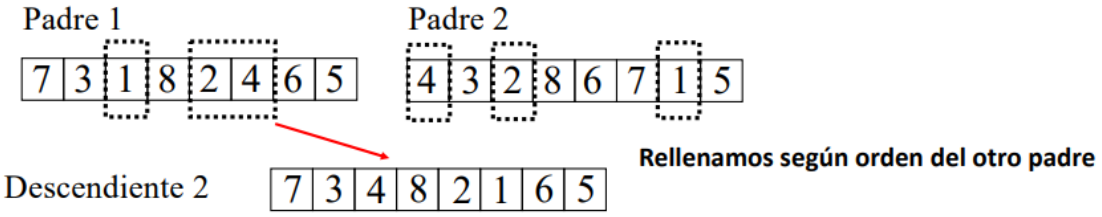


vi. Permutaciones

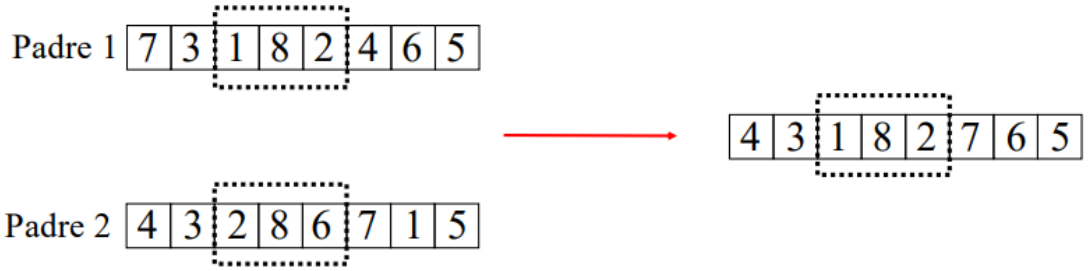
- Operador de cruce OX1



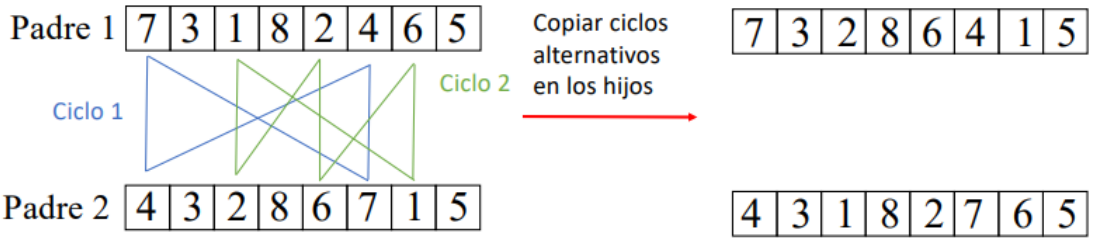
- Operador de cruce OX2. Es una modificación del OX1 en el que se escogen al azar varios genes de uno de los padres para imponer en el otro padre el orden de los elementos en las posiciones seleccionadas



- Operador de cruce PMX (Partially Mapped Crossover)



- Operador de cruce por ciclos



f. Operadores de mutación: La mutación es un proceso para explorar el espacio de búsqueda con el objetivo de descubrir áreas prometedoras lejos de los padres, siendo la única que puede introducir nuevos genes en la población.
   i. Mutación clásica/estándar

- Se selecciona uno o varios genes y se cambian sus valores
  - Un único gen

  - Múltiples genes

  - Cada gen se altera con una probabilidad p_m

ii. Mutación uniforme

- Se selecciona uno gen y su valore (entero o real) es escogido aleatoriamente entre los valores de un rango

iii. Mutación basada en intercambios y reordenación.

- Intercambios: se seleccionan dos genes y se intercambian sus valores

- Reordenación: se selecciona un subconjunto de genes y sus valores son reordenados de manera aleatoria

iv. Mutación basada en inserción.

- Selecciona aleatoriamente un gen del genotipo y lo inserta en otro lugar al azar

- Puede realizarse para un conjunto de genes

v. Mutación basada en inversión

- Selecciona un conjunto de genes del genotipo y los inserta de manera inversa (como la mutación por reordenación pero manteniendo cierto orden)

- Puede realizarse insertándola en otra posición

g. Estrategias de reemplazo: La presión selectiva se ve afectada por la forma en que los cromosomas de la población se ven reemplazados por los nuevos descendientes.
   i. Modelos generacionales: Uso de elitismo
   ii. Modelos estacionarios:

- **Reemplazar al peor de la población:** genera alta presión selectiva
- **Torneo restringido:** se reemplaza al más parecido de entre N individuos. Mantiene una cierta diversidad
- **Peor entre semejantes:** se parte de un descendiente generado y éste reemplaza al peor individuo de un conjunto de los N más parecidos (padres o individuos de la población anterior). Busca un equilibrio entre diversidad y presión selectiva
- ***Crowding* determinístico:** el hijo reemplaza a su padre más parecido. Mantiene diversidad

h. Criterio de parada

- Al alcanzar el óptimo
- Recursos limitados de CPU
  - Fijar el número máximo de generaciones
  - Fijar el número máximo de iteraciones
- Basado en la evolución
  - Después de algunas iteraciones sin mejora

7. CONCEPTO DE PRESIÓN SELECTIVA

Representa en qué grado la reproducción está dirigida por los mejores individuos.

8. CONVERGENCIA

a. Centrar la búsqueda en regiones prometedoras mediante presión selectiva.
b. La presión selectiva permite que los mejores individuos sean seleccionados para reproducirse.

9. DIVERSIDAD

a. Evita la convergencia prematura (rápida convergencia hacia zonas que no contienen el óptimo global).
b. Está asociada a las diferencias entre los individuos de la población.
c. Una falta de diversidad implica una convergencia prematura.
d. Soluciones:
   i. Reinicialización al producirse una convergencia prematura.

- Inclusión de diversidad en la evolución:
  - Diversidad con la mutación
  - Diversidad con el cruce
  - Separación espacial
  - Adaptación, auto-adaptación, metaevolución
  - Estrategias de reemplazamiento

10. ALGORITMO CHC

a. Combina una selección elitista que preserva los mejores individuos aparecidos hasta el momento con un operador de cruce que produce hijos muy diferentes a sus padres.

b. Introduce 4 componentes novedosos:

- Cruce Uniforme HUX
- Prevención de Incesto ← Aumentan la diversidad
- Reinicialización

- Selección Elitista ← Aumenta la convergencia

- **Selección Elitista.** Selecciona los N mejores cromosomas entre padres e hijos. Los N mejores elementos encontrados hasta el momento permanecerán en la población actual.
- **Cruce Uniforme (HUX).** Intercambia exactamente la mitad de los alelos que son distintos en los padres. Esto garantiza que los hijos tendrán una distancia de Hamming máxima con respecto a sus dos padres.

- **Prevención de Incesto.** Se forman N/2 parejas con los elementos de la población. Sólo se cruzan las parejas cuyos miembros difieren en un número determinado de bits (umbral de cruce).

  El umbral se inicializa a L/4 (L es la longitud del cromosoma). Si durante un ciclo no se crean descendientes mejores que los de la población anterior, al umbral de cruce se le resta 1.
- **Reinicialización.** Cuando el umbral de cruce es menor que cero, la población se reinicializa: a) usando el mejor elemento como plantilla (35% de variación aleatoria) e incluyendo una copia suya, o b) manteniendo el mejor o parte de los mejores de la población y el resto se generan aleatoriamente.
- **CHC no aplica el operador de mutación.**

TEMA 4. PROGRAMACIÓN GENÉTICA

1. CONCEPTO DE PROGRAMACIÓN GENÉTICA

Es una metaheurística poblacional cuya estructura es básicamente igual a la de un AG.

2. CARACTERÍSTICAS DE LA PROGRAMACIÓN GENÉTICA

a. Los algoritmos de GP tienen la estructura básica de otros EAs.
b. El esquema algorítmico más popular es el de estado estacionario.

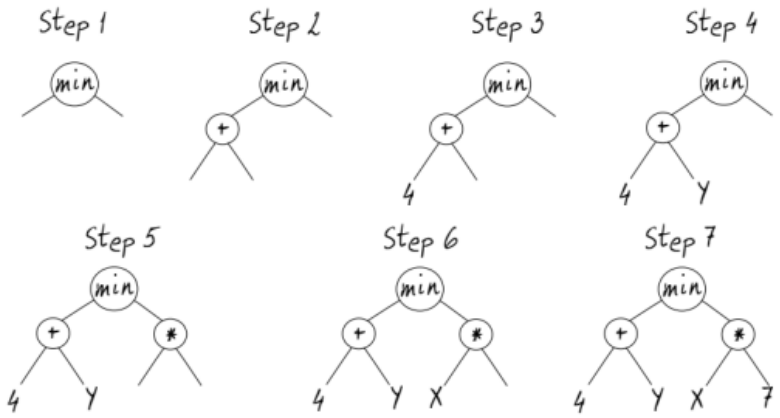
3. VARIANTES DE LA PROGRAMACIÓN GENÉTICA

a. Canonical Genetic Programming

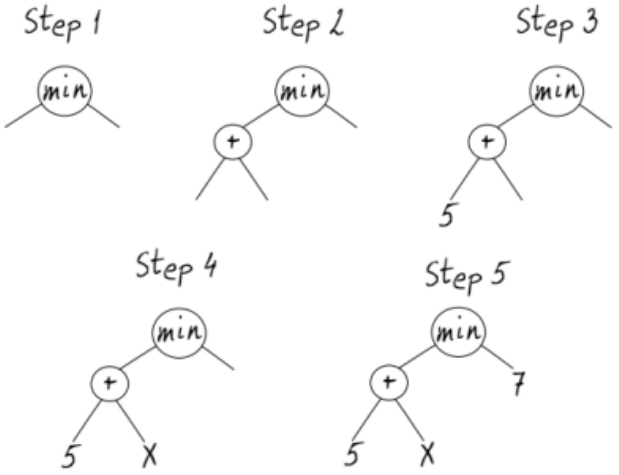
- **Individuo:**
  - Una o más expresiones expresadas como árboles
- **Nodos del árbol** de expresión:
  - Terminales o nodos hoja (variables, constantes, funciones sin argumentos)
  - Funciones o nodos internos (operadores – principalmente unarios o binarios)
- El árbol hace la doble función de genotipo y fenotipo:
  - Genotipo: experimenta transformaciones genéticas
  - Fenotipo: El recorrido del árbol genera la expresión asociada a éste

- La evaluación de los árboles consiste en ejecutar el programa representado por el árbol.
- La ejecución se hace de forma distinta según la plataforma de desarrollo:
  - La formulación original de Koza utilizaba expresiones S en el lenguaje LISP.
  - Existen implementaciones en C, C++, Java…
    - Implementaciones recursivas
    - Implementaciones basadas en uso de una pila
    - Traducción y posterior ejecución
    - …
  - La implementación influye en el coste y rendimiento del algoritmo
- La creación de la población inicial es un proceso aleatorio
- La profundidad de un árbol se calcula como el número de nodos existentes desde la raíz hasta la hoja más lejana
- Las formas más conocidas de generación son:
  - *Full Method*: genera árboles cuyas hojas están a la misma profundidad. Que los árboles tenga la misma profundidad no quiere decir que tengan la misma forma (sólo ocurre si las funciones tienen la misma aridad).
  - *Grow Method*: genera árboles de cualquier forma con la única restricción de parar al alcanzar la profundidad máxima.
  - El método *Half and Half* construye un 50% de árboles con el método *Full* y el otro 50% con el método *Grow*.
- La estrategia Ramped establece múltiples niveles de profundidad máxima

Inicialización de la población: Método Full



Inicialización de la población: Método Grow



Operadores genéticos: Cruce

- **Operador de cruce**: Se selecciona un nodo de un padre de manera aleatoria. Una vez seleccionado el punto de corte en el primer padre, se hace una elección aleatoria dirigida en el segundo padre para verificar restricciones como:
  - Cruzar con un subárbol que evalúe a un tipo de dato compatible
  - Cruzar con un subárbol de tamaño adecuado
- En programación genética, el **cruce es un operador muy disruptivo**
- Los **árboles tienden a crecer indefinidamente** si no se limita el tamaño de la operación de cruce (descendiente que no supere tamaño máximo)

- El **operador de cruce** en programación genética de dos individuos iguales no genera dos descendientes iguales como en los algoritmos genéticos. Esto provoca:
  - Dos árboles muy parecidos que están próximos a converger al óptimo no necesariamente producen descendientes adecuados
  - El algoritmo explora bien el espacio pero no lo explota bien

Operadores genéticos: Mutación

- Existen dos grandes tipos de mutación que producen una mayor o menor alteración:
  - **Un punto**: Se escoge un nodo y se cambia su valor por otro del mismo tipo
  - **Subárbol aleatorio**: Se escoge una arista y se sustituye el subárbol conectado a ella por otro generado aleatoriamente
- En programación genética, el **operador de mutación tiene menos importancia que en los algoritmos genéticos**, puesto que el operador de **cruce se basta para mantener la diversidad** (cuando la población es suficientemente grande y tiene suficiente variedad genética)

- **Permutación**: Se reemplaza un subárbol (lista de argumentos a un nodo función) por una permutación de los mismos
- **Edición**: Simplifica una expresión reemplazándola por otra más sencilla. Si se aplica sobre todos los individuos generados, es muy costosa y no se sabe si mejora la convergencia (puede reducir la diversidad). Sólo se aplica por motivos estéticos de aspecto de las expresiones
- **Encapsulación**: permite dar un nombre a un subárbol, convirtiéndolo en una nueva función que puede intervenir en otros árboles. Es equivalente a restringir los posibles puntos de cruce del árbol

b. Strongly-Typed Genetic Programming

Fundamentos

- Los individuos en STGP son similares a los individuos en CGP, es decir, son árboles de instrucciones
- Los nodos definidos en STGP están **tipados**:
  - Los argumentos que reciben (hijos) han de ser de un determinado tipo.
  - El valor que devuelven es de un tipo determinado (no tiene por qué ser el mismo que el de los argumentos de entrada
    - La función > tiene el prototipo (numérico, numérico):boolean
  - Los nodos función admiten sobrecarga, es decir, un mismo nodo puede devolver un resultado distinto según los nodos sean de un tipo o de otro:
    - Ejemplo, la función + podría tener los prototipos
      - (entero, entero): entero
      - (real, real): real
      - (vector de reales, vector de reales) : vector de reales

- La clave de STGP está en la tabla de compatibilidades que se establece al definir los nodos función y los nodos terminal.
- Esta tabla contiene, para cada nodo:
  - Número de argumentos
  - Tipo de dato para cada uno de los argumentos
  - Tipo devuelto
- Si un nodo está sobrecargado, tendrá varias entradas como esta
- A partir de esa tabla, se puede construir una segunda en la que se incluyen los nodos que pueden actuar como hijos de cada uno de los nodos existentes. Esto es lo que permite manipular los árboles para que siempre resulten correctos.

Creación de árboles y operadores genéticos

- El proceso de creación de árboles en STGP es muy parecido al de CGP. La única diferencia es que, los nodos se eligen al azar de entre los que son compatibles a un nodo dado.
- En el caso de operadores genéticos (cruce y mutación) estamos en la misma situación:
  - Para el cruce, se eligen ramas cuyos tipos devueltos sean iguales o compatibles con el lugar de destino.
  - Para la mutación, se cambian unas funciones por otras con el mismo prototipo y, cuando hay que generar un subárbol se hace usando la tabla de compatibilidad de funciones.

c. Gene Expression Programming (GEP)

- Presenta una doble codificación:
  - Genotipo lineal (parecido a los individuos de AG con codificación entera)
  - Fenotipo árbol de instrucciones (ejecutable y evaluable)

Codificación

- Los cromosomas en GEP presentan dos zonas
  - Cabeza (*head*) – Desde el extremo izquierdo hasta la posición h
  - Cola (*tail*) – Desde la posición h+1 hata la posición h+t
- Los elementos de la cabeza pueden ser funciones o terminales, pero los de la cola sólo pueden ser terminales.
- Los valores *h* y *t* se denominan *tamaño de cabeza* y *tamaño de cola*, y se definen para garantizar que la traducción del genotipo al fenotipo producirá siempre un árbol de instrucciones válido.

$$t = h\,(n_{max} - 1) + 1$$

Decodificación

Para pasar del genotipo al fenotipo se va recorriendo el cromosoma lineal, y con su contenido se van rellenando los nodos del árbol por capas: primero la raíz, segundo la capa bajo la raíz y así hasta completar el árbol

Operadores genéticos

- Existen múltiples operadores genéticos en GEP:
  - Replicación
  - Recombinación
  - Mutación
  - Transposición
  - Inversión
- Todos los operadores se aplican con una determinada probabilidad durante la fase de reproducción.

d. Grammar-Guided Genetic Programming
   Modelo GP que utiliza árboles sintácticos para representar a los individuos.
   i. Función lógica: Elementos del árbol.
   ii. Nodos terminales: Variables de la función.
   iii. Nodos no terminales: Operadores de la función.

e. Modelo GA-P

Operadores genéticos

- Al presentar los individuos una doble codificación, se pueden aplicar operadores genéticos sobre ambas partes
  - Sobre el árbol de instrucciones
  - Sobre el vector de coeficientes
  - Sobre ambos a la vez
- En los dos primeros casos, el hijo hereda la parte del padre que no ha sido alterada por el cruce y/o la mutación

TEMA 5. COLONIA DE HORMIGAS

1. PROPIEDADES DE LAS COLONIAS DE HORMIGAS

a. Eficaz en problemas de obtener el camino más corto.
b. En una intersección, el camino se decide de modo probabilístico en base al rastro de feromona.
   i. Las bifurcaciones más prometedoras van acumulando feromonas al ser recorridas por más hormigas (reclutamiento de masas).
   ii. Las bifurcaciones menos prometedoras pierden feromona por evaporación al ser visitadas por menos hormigas cada vez.

2. ALGORITMOS ACO

Probabilidad de elegir un camino

La regla que determina la probabilidad de una hormiga ***k*** de tomar una camino que va de ***r*** a ***s*** se determina como:

$$p_k(r,s) = \begin{cases} \dfrac{[\tau_{rs}]^{\alpha} \cdot [\eta_{rs}]^{\beta}}{\sum_{u \in J_k(r)} [\tau_{ru}]^{\alpha} \cdot [\eta_{ru}]^{\beta}}, & \text{si } s \in J_k(r) \\ 0, & \text{en otro caso} \end{cases}$$

- τ_{rs} es la feromona del arco *rs*
- $J_k(r)$ es el conjunto de nodos alcanzables desde *r* no visitados aún por la hormiga *k*
- η_{rs} es la información heurística del arco *rs*
- α y β son pesos que establecen un equilibrio entre la importancia de la información memorística y heurística

Actualización de la feromona

- Retroalimentación positiva: buenas soluciones con un aporte adicional de feromona. Cuanto mejor sea la solución, más feromona se aporta
- Evaporación de feromona: común para todos los rastros, eliminándose un porcentaje ($0 \leq \rho \leq 1$) de su valor actual
  - Evita un incremento ilimitado de los rastros de feromona
  - Olvida las malas decisiones tomadas
- Los arcos visitados por hormigas en la iteración actual (arcos prometedores) reciben un aporte extra de feromona y los no visitados por ninguna hormiga (poco prometedores) pierden feromona

$$\tau_{rs}(t) = (1-\rho)\cdot\tau_{rs}(t-1) + \sum_{k=1}^{m} \Delta\tau_{rs}^{k}$$

$$\Delta\tau_{rs}^{k} = \begin{cases} \dfrac{1}{C(S_k)}, & \text{si la hormiga } k \text{ ha visitado el camino } rs \\ 0, & \text{en caso contrario} \end{cases}$$

$C(S_k)$ es el coste de la solución generada por la hormiga *k*, o la longitud del circuito S_k
m es el número de hormigas

Algoritmo Sistema de Hormigas

Inicialización de parámetros (cantidad inicial de feromona)
Genera población de *m* soluciones (hormigas)
Para cada hormiga *k*
Calcula el fitness de *k*
Determina la mejor hormiga global
Actualiza las feromonas
Parar si se cumple la condición de parada

TEMA 6. PSO

1. INTRODUCCIÓN A LAS NUBES DE PARTÍCULAS

a. Es una metaheurística poblacional inspirada en el comportamiento social del vuelo de las bandadas de aves y el movimiento de los bancos de peces.
b. La población se compone de varias partículas que se mueven por el espacio de búsqueda durante la ejecución del algoritmo.
c. Este movimiento de cada partícula p depende de:
    i. Su mejor posición desde que comenzó el algoritmo (p_{best}).

ii. La mejor posición de las partículas de su entorno (l_{best}) o de toda la nube (g_{best}) desde que comenzó el algoritmo.

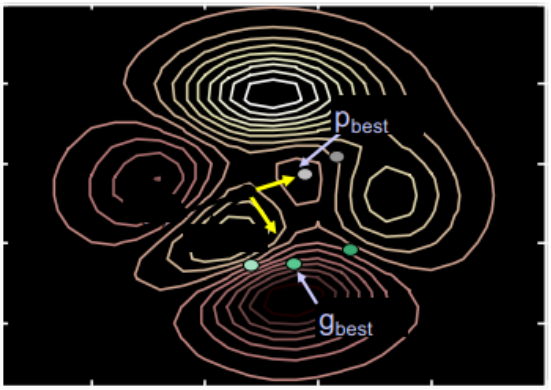
d. En cada iteración, se cambia aleatoriamente la velocidad de p para acercarla a las posiciones p_{best} y l_{best}/g_{best}.

2. CARACTERÍSTICAS ATRIBUIDAS A LAS PSO

a. Asume un intercambio de información entre los agentes de búsqueda.
b. Implementación sencilla y con pocos parámetros.
c. Convergencia rápida hacia buenas soluciones.
d. Cada partícula:
   i. Puede interactuar con un número de vecinos.
   ii. Conoce el fitness y posición de cada uno de sus vecinos.
   iii. Implementa los principios de comparar e imitar.

Cada partícula aprende ajustando su posición y velocidad:

- Parcialmente atraído a su mejor posición (p_{Best}) hasta el momento
- Parcialmente atraído a la mejor posición (g_{Best}) de la vecindad



- Movimiento de las partículas:
  - Se hace simplemente añadiendo el vector velocidad v_i al vector posición x_i para obtener un nuevo vector posición:

$$x_i^{t+1} = x_i^t + v_i^{t+1}$$

  - Una vez calculada la nueva posición de la partícula, se evalúa ésta. Si el nuevo fitness es mejor que el que la partícula tenía hasta ahora, pBestFitness, entonces:

$$p_{best,i} = x_i^{t+1}$$

  - El vector velocidad de cada partícula es modificado en cada iteración utilizando la velocidad anterior, un componente cognitivo y un componente social. El modelo matemático resultante, y que representa el corazón del algoritmo PSO, viene representado por las siguientes ecuaciones:

$$v_i^{t+1} = w \cdot v_i^t + \varphi_1 \cdot rand(0,1) \cdot \left(p_{best,i} - x_i^t\right) + \varphi_2 \cdot rand(0,1) \cdot \left(g_{best} - x_i^t\right)$$

- Vector velocidad:

$$v_i^{t+1} = w \cdot v_i^t + \varphi_1 \cdot rand(0,1) \cdot \left(p_{best,i} - x_i^t\right) + \varphi_2 \cdot rand(0,1) \cdot \left(g_{best} - x_i^t\right)$$

componente cognitivo **componente social**

x_i^t es el vector posición de la partícula *i* en la iteración *t*

v_i^t es el vector velocidad de la partícula *i* en la iteración *t*

ω es el vector de inercia

φ_1 φ_2 son los pesos que controlan los componentes cognitivo y social

rand es una función que genera un valor aleatorio entre 0 y 1

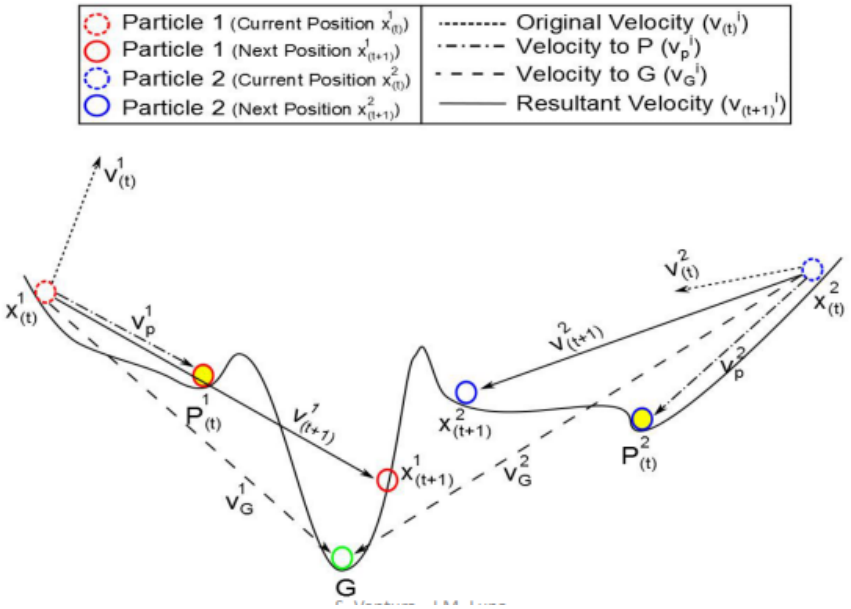
$p_{best,i}$ es la mejor posición encontrada por la partícula *i* hasta el momento

g_{best} representa la posición de la partícula con la mejor solución

- **Componente cognitivo**: indica la decisión que tomará la partícula y depende de su propia experiencia. Representa la distancia entre la posición actual y la mejor conocida por esa partícula

- **Componente social**: apunta la decisión que tomará la partícula en base a la influencia del resto de partículas que componen la nube. Representa la distancia entre la posición actual y la mejor posición encontrada por vecindario

Movimiento de dos partículas en una dimensión



3. TIPOS DE ALGORITMOS

- Según la importancia de los pesos cognitivo y social:
  - **Completo**, si $\varphi_1 > 0$ *y* $\varphi_2 > 0$ (Tanto el componente cognitivo como el social intervienen en el movimiento)
  - **Cognitivo**, si $\varphi_1 > 0$ y $\varphi_2 = 0$ (Sólo interviene el componente cognitivo en el movimiento)
  - **Social**, si $\varphi_1 = 0$ y $\varphi_2 > 0$ (Sólo interviene el componente social en el movimiento)
  - **Social exclusivo**, si $\varphi_1 = 0$, $\varphi_2 > 0$, y $p_{best,i} \neq g_{best}$ (La posición de la partícula en sí no puede ser la mejor de su entorno)

- Según el tipo de vecindario utilizado (cantidad y posición de las partículas que intervienen en el cálculo de la distancia en la componente social):
  - **Local**. Calcula la distancia entre la posición actual de la partícula y la posición de la mejor partícula perteneciente al entorno local de aquella. El entorno local consiste en las partículas inmediatamente cercanas
  - **Global**. La distancia se obtiene entre la posición actual de la partícula y la posición de la mejor partícula considerando la nube completa

PSO Clásico

t=0
Nube ← Inicializar Nube de Partículas
Mientras no se alcance la condición de parada **hacer**
t=t+1
Para i = 1 hasta tamaño (Nube) **hacer**
Evaluar cada partícula x_i^t de la Nube
Si fitness de x_i^t es mejor que fitness de $p_{best,i}$ **entonces**
$p_{best,i} \leftarrow x_i^t$
fitness $p_{best,i} \leftarrow fitness\ x_i^t$
Fin Si
Si fitness de x_i^t es mejor que fitness de g_{best} **entonces**
$g_{best} \leftarrow x_i^t$
fitness $g_{best} \leftarrow$ fitness _ x_i^t
Fin Si
Fin Para
Para i = 1 hasta tamaño (Nube) **hacer**
Calcular la velocidad v_i de x_i, en base a los valores x_i, mejorpos$_i$ y mejorpos
Calcular la nueva posición de x_i, de su valor actual y v_i
Fin Para
Fin Mientras
Salida: Devuelve la mejor solución encontrada.

a. Ejemplo de aplicación: Minimización de funciones.

TEMA 7. OTROS PROBLEMAS

1. PROBLEMAS MULTIMODALES

a. Son problemas que tienen múltiples óptimos locales/globales (múltiples soluciones al problema).

b. Evolución en problemas multimodales

i. Se comienza con una población inicial que proporciona un muestreo aleatorio del espacio de soluciones.

ii. **Deriva genética**: El proceso evolutivo suele provocar la convergencia de toda la población a una zona restringida del espacio de búsqueda, abandonando la exploración del resto de óptimos locales.

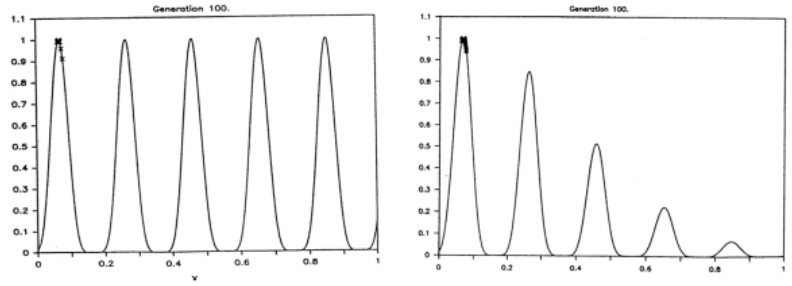
iii. El objetivo principal es preservar la diversidad en la población, permitiendo la búsqueda simultánea en diferentes áreas controlando la

competencia de las soluciones dentro y fuera de las áreas.

iv. **Uso de algoritmos de nichos o multimodales**: Los algoritmos genéticos evolucionan una población que permite obtener soluciones en diferentes zonas del espacio de búsqueda (nichos).

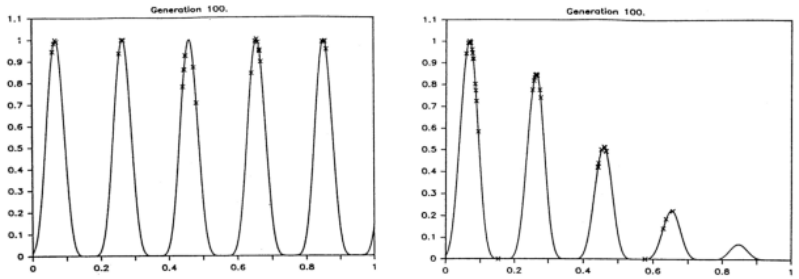
c. Evolución sin nichos vs Evolución con nichos

- Evolución **sin nichos y sin mutación** sobre un problema con varios óptimos globales (izquierda) y varios óptimos locales (derecha)



- Se produce una convergencia hacia cualquier óptimo debido a la **deriva genética**

- Evolución **con nichos y sin mutación** sobre un problema con varios óptimos globales (izquierda) y varios óptimos locales (derecha)



- Se produce una convergencia hacia varios óptimos mediante técnicas de nichos

d. Clasificación de los algoritmos de nichos según la formación de nichos

- **Espaciales**: Formación de diferentes nichos en las poblaciones de una misma ejecución del algoritmo genético. Dos ejemplos:
  - *Fitness Sharing* (Penalización de la calidad)
  - *Clearing* (Limpieza en la población)

- **Temporales**: Formación de diferentes nichos a lo largo de diferentes ejecuciones del algoritmo genético

AGs Multimodales Espaciales

Fitness Sharing (método de proporción)

Pretende formar subconjuntos de elementos vecinos en la población llamados **nichos**, asociando cada uno de ellos con óptimo (multimodalidad).

Proceso: Modifica la calidad de los individuos que pertenecen a zonas densamente pobladas.

- Antes del proceso de selección se analizan los individuos de la población calculando su valor de adaptación modificado (f*) en función de su cercanía al resto de individuos que componen la población.
- Con esta nueva adaptación (f*) se realiza el proceso de selección.

Fitness Sharing (método de proporción): Formulación y parámetros

$$f_i^* = \frac{f_i}{\sum_{j=1}^{N} sh(d(i,j))}$$

$$sh(d(i,j)) = \begin{cases} 1 - \left(\frac{d(i,j)}{\sigma_{share}}\right)^{\alpha} & si\ d(i,j) < \sigma_{share} \\ 0 & resto \end{cases}$$

La función f decrece en razón al número de elementos pertenecientes a su nicho contenidos en la población en un momento dado, sin embargo, crece en razón al valor de su función de evaluación. Cuando el nicho tiene un único cromosoma la función no se modifica ya que Sh(d(i,i)) = 1.*

$d(i, j)$	Distancia entre los individuos i y j.
σ_{share}	Radio del nicho: determina la pertenencia o no al nicho.
α	Regulador de la pendiente de la función de sharing. Valores comúnmente utilizados (1) y (2).

Clearing (aclarado)

Proceso: La selección se realiza únicamente sobre los individuos predominantes en cada nicho.

- Antes del proceso de selección se clasifica la población según la adaptación de forma decreciente.
- Se coge al primer individuo (el mejor) y de forma descendente se compara con el resto de la población. Aquellos individuos que están dentro de su radio de nicho (individuos dominados) son eliminados.
- El proceso continúa con el segundo individuo de la clasificación que aún no haya sido eliminado, y eliminará los individuos dominados por el.
- El proceso terminará cuando tengamos los elementos predominantes de cada nicho y con ellos se realizará la selección.

Proceso:

```
Ordenar P de mejor a peor
for i=0 to N-1
{
    if (Fitness (P[i])>0)
    {
        NumGanadores=1
        for j=i+1 to N-1
                if (Fitness (P[j])>0) and (Distancia(P[i],P[j])<σ))
                {
                        if (NumGanadores<Kappa)
                                NumGanadores ++
                        else
                                Fitness(P[j])=0
                }
    }
}
```

Parámetros:

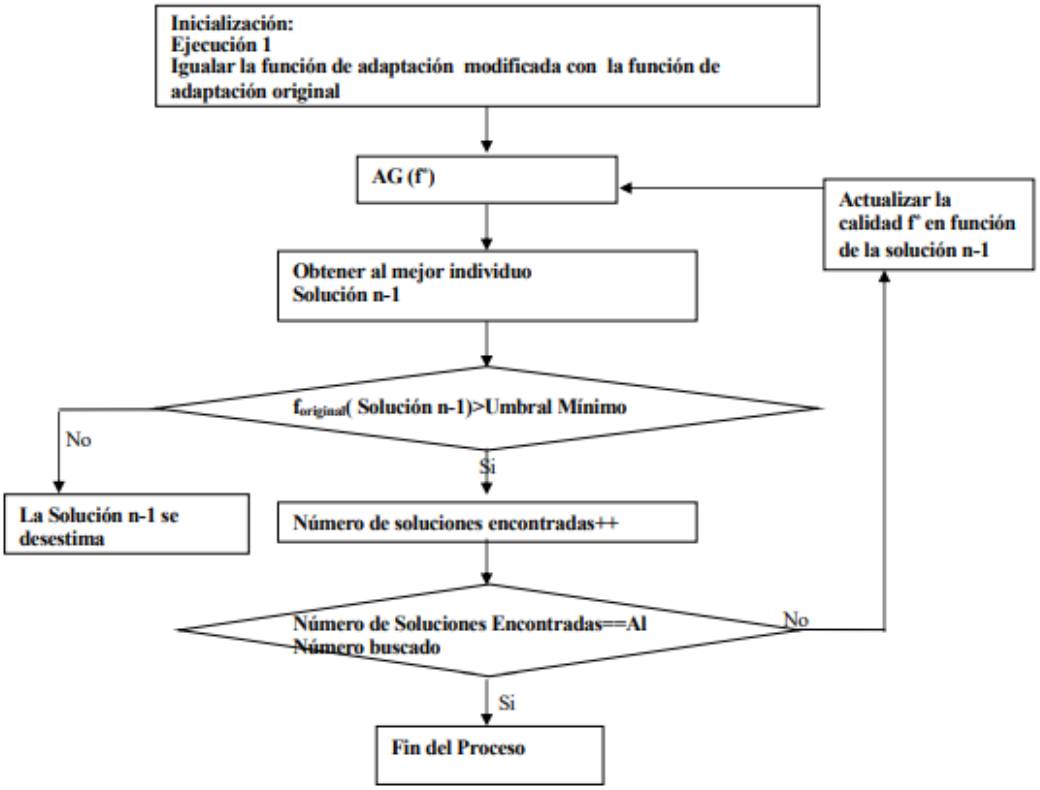
σ Radio de nicho

Kappa. Número de individuos que se mantienen por nicho

AGs Multimodales Temporales

Sequential (Nichos secuenciales)

- Consiste en la ejecución secuencial de AGs básicos de forma dependiente.
- Con el primer AG se obtiene una solución, si su calidad está por encima de un umbral mínimo se guarda como solución del problema y se incrementa el contador de soluciones halladas.
- Con esta solución, se modifica la adaptación de los individuos del siguiente AG, penalizando de esta forma las zonas ya exploradas en el AG anterior. Con esta nueva ejecución, se obtiene otra solución.
- Para las siguientes ejecuciones se modificará la adaptación teniendo en cuenta todas las soluciones encontradas en pasos previos.
- El proceso termina cuando tengamos las soluciones deseadas.



$$f_n^*(x) = f_{n-1}^*(x) G(x, s_{n-1})$$

$$G(x, s_{n-1}) = \begin{cases} \left(\frac{d(x, s_{n-1})}{\sigma_{share}}\right)^{\alpha} & si \;\; d(x, s_{n-1}) < \sigma_{share} \\ 1 & en\ otro\ caso \end{cases}$$

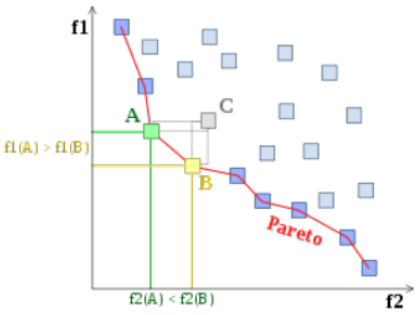
s_{n-1}: Solución encontrada en la ejecución n-1

x: Individuo de la ejecución n

2. PROBLEMAS MULTIOBJETIVO

Muchos problemas reales se caracterizan por la existencia de **múltiples medidas de actuación**, las cuales deberían ser optimizadas, o al menos ser satisfechas simultáneamente.

- Diseñar un dispositivo electrónico, donde se desea maximizar el desempeño y minimizar el coste de fabricación
- Aire acondicionado, donde se desea maximizar el confort y minimizar el consumo



Dominancia de Pareto

- Un problema multiobjetivo consiste en:

$$\text{Max o Min } z = f(x) = (f_1(x), f_2(x), ..., f_n(x))$$

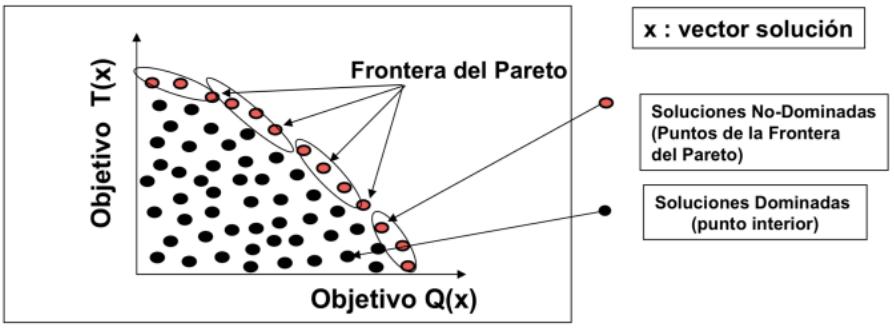
- **Soluciones Pareto-optimales o no dominadas:** Se dice que un vector ***a*** domina a otro ***b*** si y sólo si:

$$\forall i \in \{1,2,...,n\} \mid f_i(a) \geq f_i(b) \wedge \exists\, j \in \{1,2,...,n\} \mid f_i(a) > f_i(b)$$

Es decir, una solución domina a otra si es mejor o igual en todos los objetivos y al menos mejor en uno de ellos. Todos los vectores que no son dominados por ningún otro vector se llaman **Pareto-optimales o no-dominados**

Frontera de Pareto

- No suele existir una única solución *optimal*, sino un conjunto (a veces infinito) de soluciones no dominadas que forma la **Frontera del Pareto**



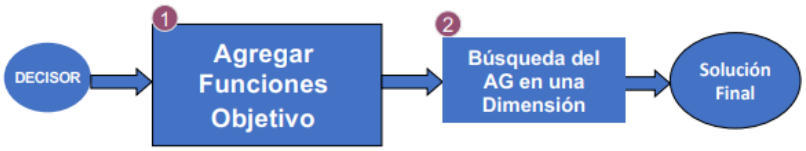
¿Qué necesitamos para resolver un problema multi-objetivo?:

- Un método de búsqueda basado en los múltiples objetivos.
- Una política de equilibrio entre los objetivos.
- Un orden para este proceso de optimización.

Podemos proceder de dos formas :

- Opción 1: Agregar + Buscar
- Opción 2: Buscar + Agregar

Algoritmos Multi-Objetivo: Agregación + Búsqueda

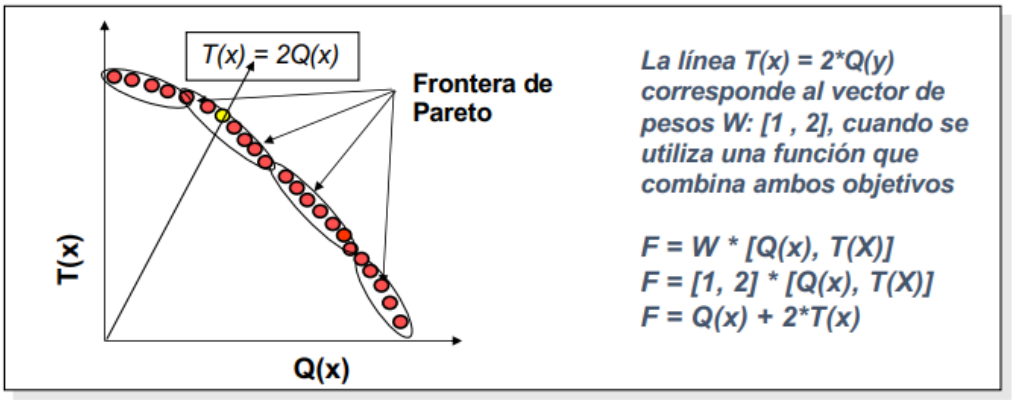


Primero se agregan los objetivos dando lugar a una función de fitness para el AG que se aplica a continuación.

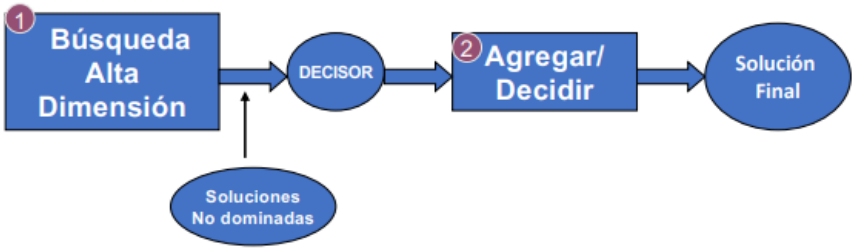
Algoritmos Multiobjetivo que usan Pesos

- La agregación de los objetivos conduce a la obtención de un único punto de equilibrio en la frontera.
- Ejemplo: *[Max Q(x), Max T(x)]*

  *Dar a T(x) dos veces la importancia de Q(x), ej: T(x) = 2*Q(x)*



Algoritmos Multi-Objetivo: Búsqueda + Agregación



Nota: Se puede considerar una tercera posibilidad híbrida, combinando búsqueda en alta dimensión con búsquedas en dimensiones menores vía agregación parcial de objetivos, como modelos interactivos.

Modelos que generan soluciones no dominadas: MOGA

MOGA: Multi-objective Optimization GA *(Fonseca & Fleming 1993)*

C.M. Fonseca, P.J. Fleming, Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. S. Forrest (Ed.), Proc. 5th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, 1993, 416-423.

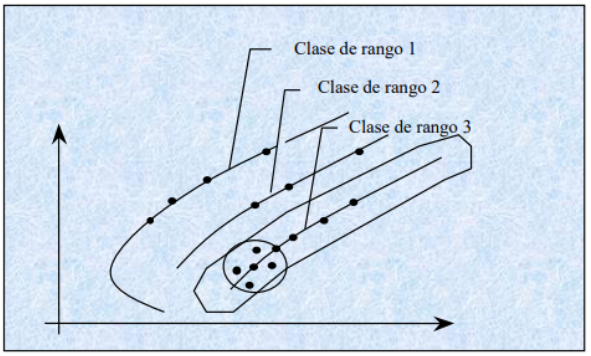
A cada individuo de la población se le asigna un rango de acuerdo al cual será ordenado para la selección.

El rango se asigna según un criterio de no dominancia.

si x_i *es no dominado* ***entonces***
  $rango(x_i) = 1$
sino
  $rango(x_i) = 1 +$ *(no. de individuos que lo dominan)*
fin si

Una vez calculado el rango de los individuos de la población se siguen los siguientes pasos:

1. La población se ordena de menor a mayor de acuerdo al rango que se le ha asignado a cada individuo.
2. Se asigna el valor de adaptación para cada individuo por interpolación desde el mejor (rango 1) hasta el peor.
3. Se promedia la adaptación de los individuos con el mismo rango, para que tengan el mismo valor de adaptación.



Modelos que generan soluciones no dominadas: NPGA

- NPGA: Niched Pareto GA (*Horn & Nafpliotis, 1993*)

J. Horn, N. Nafpliotis. Multiobjective Optimization Using the Niched Pareto Genetic Algorithms. IlliGAL Report 93005, University of Illinois, Urbana, Champaign, July 1993.

Este algoritmo se basa en **la combinación de la selección de torneo y el concepto de dominancia de Pareto.**

- Dados dos individuos que compiten, se selecciona un subconjunto aleatorio de la población de tamaño t_{dom}.
- Si un individuo es dominado por cualquier miembro del conjunto y el otro no, entonces este último se considera el ganador del torneo.
- Si ambos individuos son dominados, el resultado del torneo se decide por el método de proporción (el individuo con menos semejantes en su nicho es el seleccionado - la técnica de *sharing* se aplica a nivel de objetivos).

Modelos que generan soluciones no dominadas: NSGA

- NSGA: Non-dominated Sorting GA (*Srinivas & Deb, 1995*)

N. Srinivas, K. Deb, Multiobjetive Optimization Using Nondominated Sorting in Genetic Algorithms. Evolutionary Computation 2 (1995) 221-248.

Este algoritmo se basa en:

- Una ordenación de la población según el criterio de no-dominancia (distinto de MOGA), mediante fronteras de no-dominancia.
- El uso de una técnica de proporción de nichos (*sharing*) aplicada sobre los valores de las variables de decisión de cada individuo.

El algoritmo actúa extrayendo frentes de individuos progresivamente, a los que se asigna un valor de adaptación menor que el del frente anterior.

1. En el primer frente se toman los individuos no dominados, a los que se asigna un valor hipotético alto como valor de adaptación.
2. Estos individuos se penalizan según un criterio de proporción en el fenotipo (*sharing* en las variables) con α=2.
3. Los individuos anteriores son ignorados temporalmente para procesar el resto de la población de igual forma, identificando el segundo conjunto de individuos no dominados (entre los dominados).
   A este segundo conjunto de individuos se le asigna un valor hipotético más pequeño que el mínimo valor alcanzado por el conjunto anterior tras la aplicación del método de proporción.
4. El proceso continua hasta que toda la población se clasifica en frentes.

Elitismo en la Búsqueda Evolutiva Multiobjetivo. Conjunto Elite

La segunda generación de MOEAs usa una **población externa**, donde se almacenan soluciones no-dominadas encontradas a lo largo de la búsqueda. Esto permite al algoritmo cubrir de un modo más adecuado el Frente del Pareto.

Este conjunto de soluciones no dominadas se suele llamar "conjunto elite", P_e, con tamaño N_e. Su uso lleva asociadas dos cuestiones:

Población → Conjunto elite.

¿Qué soluciones de P se mantienen en P_e?

Conjunto elite → Población.

¿Cómo y cuando los elementos de P_e se reinsertan en P?

Modelo genérico

Modelo Genérico de Algoritmo Evolutivo Multiobjetivo Elitista

$t \leftarrow 0$

$(A^0, B^0, p_e^0) \leftarrow$ inicializar()

Mientras terminar(A^t, B^t, t) = falso **Hacer**

$t \leftarrow t + 1$

$A^t \leftarrow$ truncar(actualizar(A^{t-1}, B^{t-1}))

$p_e^t \leftarrow$ adaptar(A^t, B^{t-1} , p_e^{t-1})

$B^t \leftarrow$ operadores(selección(evaluación ($A^t, B^{t-1}\ p_e^t$)))

Fin Mientras

A^t : población élite B^t : población

p_e^t := intensidad del elitismo en la generación t.

Algoritmo SPEA

Paso 1. Generar la población inicial P y el conjunto P_e vacío.

Paso 2. Copiar las soluciones no dominadas de P en P_e.

Paso 3. Quitar en P^e aquellas soluciones dominadas por otras.

Paso 4. Si $|P_e| > N_e$, entonces reducir el conjunto a tamaño N^e mediante técnicas de *clustering*.

Paso 5. Calcula el fitness de los individuos de $P'=P+P_e$.

Paso 6. Seleccionar N individuos a partir de P' (mediante torneo binario).

Paso 7. Aplicar cruce y mutación.

Paso 8. Volver al Paso 2 si no se ha alcanzado el máximo número de iteraciones.