Inteligencia Artificial

Tema 2: Metaheurísticas para Optimización

1. Optimización

En un conjunto O, existe una función F : O --> R tal que F cuantifica como de bueno es cada elemento de O.

Optimizar es encontrar un elemento X perteneciente a O con mejor valoración F(X). Si se busca el de menor valoración se dice minimización; si se busca el de mayor valoración, maximización. El problema computacional surge cuando el conjunto O es enormemente grande; por ello, no nos vale un simple algoritmo de fuerza bruta que recorra todo el conjunto O, tenemos que recurrir a heurísticas.

Una heurística es una técnica para resolver (de forma eficiente) un problema a partir de conocimiento específico sobre el mismo. Usualmente suponen soluciones aproximadas. Vamos a ver las siguientes técnicas:

a. Búsqueda en escalada
b. Enfriamiento simulado
c. Algoritmos genéticos
d. PSO: Particle Swarm Optimization

La mayoría de técnicas se basan en la "mejora iterativa": empezar con uno (o varios) candidatos y mejorar su calidad paso a paso.

Representación de un problema de optimización

a. Elección de una representación para los estados (estructura)
b. Función de valoración F (maximizar o minimizar)
c. Función que genera el estado inicial
d. Función que genera el estado sucesor (genera un estado sucesor dado un estado)

Búsqueda en escalada

Idea principal: aplicar una mejora iterativa guiada por la heurística y aleatoriedad de la función que genera sucesores. No se permite recuperarse de un camino erróneo (una vez tomado un camino no se puede volver).

Versión para minimizar (análogo para maximizar)

```
FUNCION BUSQUEDA-EN-ESCALADA()
1. Hacer ACTUAL igual GENERA-ESTADO-INICIAL() y
         VALOR-ACTUAL igual a F-VALORACIÓN(ACTUAL).
2. Hacer VECINO igual a GENERA-SUCESOR(ACTUAL) y
         VALOR-VECINO igual a F-VALORACIÓN(VECINO)
3. Mientras VALOR-VECINO sea menor que VALOR-ACTUAL,
   3.1 Hacer ACTUAL igual a VECINO
        y VALOR-ACTUAL igual a VALOR-VECINO.
   3.2 Hacer VECINO igual a GENERA-SUCESOR(ACTUAL) y
        VALOR-VECINO igual a F-VALORACIÓN(VECINO)
3. Devolver ACTUAL y VALOR-ACTUAL
```

Ejemplo: problema del viajante resuelto mediante búsqueda en escalada: dada una lista de ciudades se quiere pasar por todas recorriendo la menor distancia posible.

```
>>> p_andalucia=
    Viajante_BL(andalucia.keys(),
                lambda x,y: distancia_euc2D(x,y,andalucia))
>>> escalada(pa)
(['malaga', 'sevilla', 'huelva', 'cadiz',
  'almeria', 'cordoba', 'granada', 'jaen'],
 1276.4491417672511)
>>> escalada(pa)
(['jaen', 'almeria', 'malaga', 'huelva',
  'granada', 'cadiz', 'sevilla','cordoba'],
 1448.5485463804778)
>>> escalada(pa)
(['jaen', 'cordoba', 'huelva', 'cadiz',
  'granada', 'almeria', 'malaga', 'sevilla'],
 1318.7016090503269)
```

No se garantiza encontrar el óptimo. Su eficacia depende, en gran medida, de la función que genera el sucesor.

Búsqueda en escalada con reinicio aleatorio

```
FUNCION ESCALADA-CON-REINICIO-ALEATORIO(ITERACIONES)
1. Hacer MEJOR-ESTADO igual GENERA-ESTADO-INICIAL() y
         MEJOR-VALOR igual a F-VALORACIÓN(MEJOR-ESTADO)
2. Hacer un número de veces igual a ITERACIONES:
   2.1 Realizar una escalada aleatoria.
   2.2 Si el estado obtenido tiene un valor mejor
        que MEJOR-VALOR, actualizar MEJOR-ESTADO
        y MEJOR-VALOR con el nuevo estado y valor
        obtenido.
3. Devolver MEJOR-ESTADO y MEJOR-VALOR
```

Ejemplo del problema del viajante usando búsqueda en escalada con reinicio aleatorio:

```
>>> escalada_con_reinicio_aleatorio(p_andalucia, 50)
(['malaga', 'sevilla', 'huelva', 'cadiz',
  'cordoba', 'jaen', 'almeria', 'granada'],
 1002.9799545640491)
>>> escalada_con_reinicio_aleatorio(p_andalucia, 100)
(['sevilla', 'huelva', 'cadiz', 'cordoba',
  'granada', 'jaen', 'almeria', 'malaga'],
 1085.637600224146)
>>> escalada_con_reinicio_aleatorio(p_andalucia, 300)
(['cadiz', 'huelva', 'sevilla', 'cordoba',
  'jaen', 'almeria', 'granada', 'malaga'],
 929.9255755927754)
```

La solución óptima es la última.

Enfriamiento simulado

Idea principal: aceptar probabilísticamente estados “peores”. La probabilidad de que un estado peor sea aceptado varía en función de la función de valoración. Permitimos de esta manera salir de “óptimos locales” sin perder el “óptimo global”.

Este algoritmo está basado en un proceso físico-químico de enfriamiento de metales. La generación del estado inicial y de los estados “vecinos” se basa en aleatoriedad y heurística. El criterio de parada es obtener un valor suficientemente bueno dado una función e valoración, o en nuestro caso, llegar a un nº fijo de iteraciones totales.

```
FUNCION ENFRIAMIENTO-SIMULADO(T-INICIAL,FACTOR-DESCENSO,
                              N-ENFRIAMIENTOS,N-ITERACIONES)

1. Crear las siguientes variables locales:
   1.1 TEMPERATURA (para almacenar la temperatura actual),
       inicialmente con valor T-INICIAL.
   1.2 ACTUAL (para almacenar el estado actual), cuyo valor
       inicial es GENERA-ESTADO-INICIAL().
   1.3 VALOR-ACTUAL igual a F-VALORACIÓN(ACTUAL)
   1.4 MEJOR (para almacenar el mejor estado
       encontrado hasta el momento), inicialmente ACTUAL.
   1.5 VALOR-MEJOR (para almacenar el valor de MEJOR),
       inicialmente igual a VALOR-ACTUAL
2. Iterar un número de veces igual a N-ENFRIAMIENTOS:

   2.1 Iterar un número de veces igual a N-ITERACIONES:

   2.1.1 Crear las siguientes variables locales:
   2.1.1.1 CANDIDATA, una solución vecina de ACTUAL,
           generada por GENERA-SUCESOR.
   2.1.1.2 VALOR-CANDIDATA, el valor de CANDIDATA.
   2.1.1.3 INCREMENTO, la diferencia entre VALOR-CANDIDATA y
           VALOR-ACTUAL
   2.1.2 Cuando INCREMENTO es negativo, o se acepta
         probabilísticamente la solución candidata,
         hacer ACTUAL igual a VECINA
         y VALOR-ACTUAL igual a VALOR-VECINA.
   2.1.3 Si VALOR-ACTUAL es mejor que VALOR-MEJOR,
         actualizar MEJOR con ACTUAL
         y VALOR-MEJOR con VALOR-ACTUAL.

   2.2  Disminuir TEMPERATURA usando FACTOR-DESCENSO

3. Devolver MEJOR y VALOR-MEJOR
```

Ejemplo del problema del viajante usando enfriamiento simulado:

```
>>> enfriamiento_simulado(pa, 5, 0.95, 100, 100)
(['malaga', 'cadiz', 'huelva', 'sevilla',
  'cordoba', 'jaen', 'almeria', 'granada'],
 929.9255755927754)
```

Algoritmos genéticos: evolución natural

Idea principal: optimización basada en los procesos evolutivos de la naturaleza: de algo menor a algo más grande. No existe un único algoritmo genético. Un primer paso es la representación de los estados del problema como individuos de una población:

a. Genes (lo más básico)
b. Cromosomas (conjuntos de genes)
c. Población (conjuntos de cromosomas)
d. Etc.
e. Se trata de evitar los "óptimos locales" manejando poblaciones de candidatos en lugar de un único candidato. Esta población "evoluciona" a lo largo de "generaciones" (iteraciones).
f. Definimos como "Bondad de los individuos" al valor de la función de valoración (fitness).

· Evolución de las generaciones: es común la semejanza con aleatoriedad:

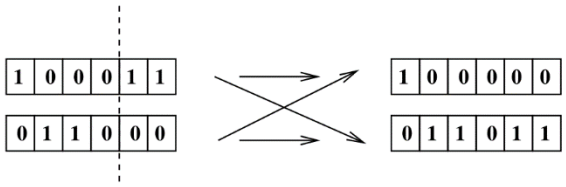
a. Operadores de variación:
    a. Cruces --> combinaciones de estructuras (padres que dan lugar a hijos).
    b. Mutaciones --> cambios de algunos genes.
b. Mecanismos de selección:
    a. Sirven para decidir que cromosomas se combinan para dar lugar a la siguiente generación.
    b. En general, debe emplearse un método de selección por mejor valoración.

```
INICIAR población
EVALUAR cada individuo de la población

Repetir hasta CONDICIÓN_DE_TERMINACIÓN
   SELECCIONAR padres
   COMBINAR pares de padres
   MUTAR hijos resultantes
   EVALUAR nuevos individuos
   SELECCIONAR individuos para la siguiente generación

Devolver el mejor de la última generación
```

(Esquema general de un algoritmo genético)

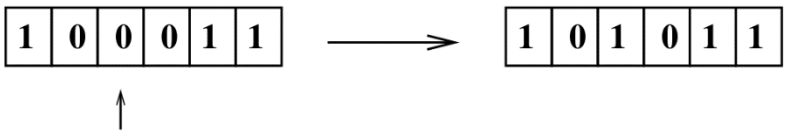
· <u>Combinación de individuos</u>:

a. Operadores que combinan la información de los padres para obtener nuevos hijos.
b. Cruce en un punto: elige un punto aleatoriamente:



c. También: cruce multipunto, uniforme, etc.

· Mutación de individuos: basadas en la aleatoriedad:



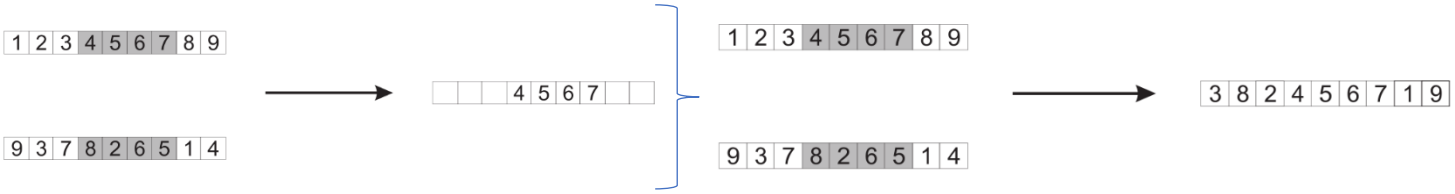
·Permutaciones: variante de la mutación: si un cromosoma es una permutación de genes, es necesario disponer de operadores de mutación y cruce:

1. Mutación por intercambio:

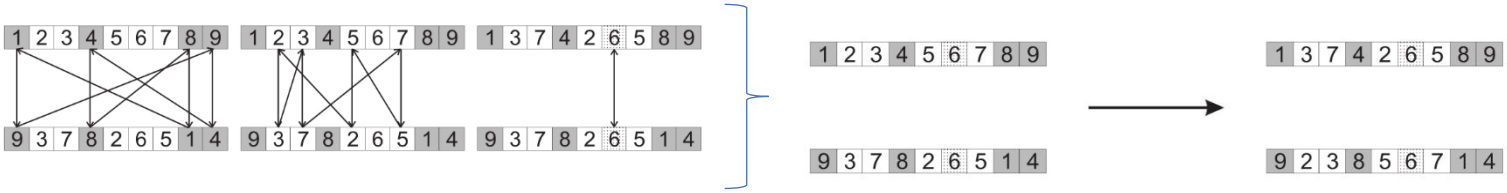
2. Mutación por inserción:

3. Mutación por mezcla:

4. Cruces:
   a. Cruce basado en orden: se eligen dos puntos aleatorios del primer padre y copia el segmento entre ambos puntos en el primer hijo. A partir del segundo punto de corte del segundo padre, copia los genes no usados en el primer hijo en el mismo orden.



   b. Cruce basado en ciclos: dividimos la permutación en ciclos y alternamos los ciclos de cada padre. Se toma la 1º posición nueva en el padre 1, buscamos el gen en la misma posición del padre 2, vamos a la posición con el mismo gen en el padre 1 y añadimos este gen al ciclo; luego repetimos los pasos hasta llegar al primer gen del padre 2.



· Mecanismos de selección: necesario un método para elegir aquellos que van a ser usados como padres y que hijos pasan a la siguiente generación. La selección debe favorecer a los mejores, permitir la diversidad y añadir una componente aleatoria (normalmente).

Métodos:

a. Proporcional a su valoración: seleccionar aleatoriamente, pero de forma que cada individuo tenga una probabilidad de ser seleccionado proporcional a su valoración respecto a las valoraciones del total de la población. Los mejores individuos se seleccionan con mayor frecuencia. La probabilidad de que cada individuo i sea seleccionado es:

$$P(i) = \frac{F(i)}{\sum_{j=1}^{n} F(j)}$$

Este método solo es válido para resolver problemas de maximización. Para resolver problemas de minimización habría que cambiar la función de "fitness". Hay una variante, llamada selección por ranking, donde la probabilidad es proporcional a la posición del individuo en la población.

b. Selección por torneo: para cada elección, se extrae aleatoriamente k individuos y se selecciona el mejor. No dependemos de la magnitud de la función de valoración y sirve tanto para maximización como minimización, pero cuanto mayor k, mayor complejidad.

c. Selección elitista: escogemos un porcentaje de los mejores. Para añadir diversidad, de entre el resto, seleccionamos algunos aleatoriamente.

· Otras componentes de los algoritmos genéticos:

a. Población inicial: usualmente, se parte de N individuos seleccionados de forma aleatoria.
b. Terminación del algoritmo: se para tras completar un nº dado de generaciones o hasta un valor prefijado de la función de valoración.
c. Diversos parámetros: tamaño de la población (cantidad de individuos), proporción de padres y probabilidades de cruce y/o mutación.

```
t := 0
Inicia-Población P(t)
Evalúa-Población P(t)

Mientras t < N-Generaciones hacer
   P1   := Selección por torneo de (1-r)·p  individuos de P(t)
   P2   := Selección por torneo de (r·p) individuos de P(t)
   P3   := Cruza P2
   P4   := Union de P1 y P3
   P(t+1) := Muta P4
   Evalua-Población P(t+1)
   t:= t+1
Fin-Mientras

Devolver el mejor de P(t)
```

(Ejemplo de selección por torneo en algoritmo genético, r es la proporción de padres)

```
t := 0
Inicia-Población P(t)
Evalúa-Población P(t)

Mientras t < N-Generaciones hacer
   P1   := Selecciona-Mejores P(t)
   P2   := Selecciona-aleatorio (P(t) - P1)
   P3   := Cruza (P1 U P2)
   P4   := Muta P3
   Evalua-Población P4
   P(t+1) := Selecciona-Mejores P4,P(t)
   t:= t+1
Fin-Mientras

Devolver el mejor de P(t)
```

(Ejemplo de selección elitista en algoritmo genético, se toman los mejores entre los hijos y los individuos originales)

Optimización de enjambre de partículas (PSO)

También basado en la naturaleza, concretamente, en el vuelo de las bandadas de pájaros. Se enmarca en el modelo conocido como "Inteligencia de Enjambre" (Swarm intelligence) que estudia el comportamiento colectivo de sistemas descentralizados auto-organizativos.

Un sistema PSO consta de N partículas que ocupan una posición en R^m y que lleva asociado un vector de velocidad, junto con una función fitness que asocia un valor a cada posición del sistema. El objetivo es encontrar la posición donde la función fitness toma el valor mínimo o máximo.

La posición y velocidad inicial se establece de forma aleatoria. Todas las partículas actualizan su posición en paralelo y se rigen por las mismas reglas. Para cada partícula, su posición en el instante t + 1 depende de su posición en el instante t más la velocidad de la partícula en el instante t + 1.

$$\vec{x}_i^{t+1} = \vec{x}_i^{t} + \vec{v}_i^{t+1}$$

Para conocer la nueva posición $\vec{x}_i^{t+1}$ de la partícula i, hay que determinar $\vec{v}_i^{t+1}$ Esa velocidad depende de:

a. La velocidad que lleva la partícula en t (inercia).
b. La mejor posición que haya obtenido la partícula i en los pasos previos.
c. La mejor posición que haya obtenido cualquier partícula en los pasos previos.

Para el cálculo de la velocidad necesitamos tres parámetros,

1. W (que pertenece a R) que representa la inercia.
2. C1 que representa la influencia individual.
3. C2 que representa la influencia grupal.

y dos valores aleatorios R1 y R2 (pertenecientes a [0,1]). De forma que:

$$\vec{v}_i^{t+1} = W \cdot \vec{v}_i^t + c_1 \cdot r_1 \cdot (\vec{m}_i^t - \vec{x}_i^t) + c_2 \cdot r_2 \cdot (\vec{best}^t - \vec{x}_i^t)$$

- $\vec{m}_i^t$ es la mejor posición de la partícula i en los pasos $0, \ldots, t$
- $\vec{best}^t$ es la mejor posición de cualquier partícula i en los pasos $0, \ldots, t$