

Tema 2: Metaheurísticas para Optimización

José Luis Ruiz Reina
Miguel A. Gutiérrez Naranjo

Departamento de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Inteligencia Artificial

Índice

Problemas de optimización y metaheurísticas

Escalada y enfriamiento simulado

Algoritmos genéticos

PSO

Un ejemplo clásico: problema del viajante

- Con 8 ciudades, existen 2520 posibles rutas (es abordable).
- Con 100 ciudades, hay **(99)!/2** posibles rutas
 - Más de **10¹⁵⁵** posibles rutas candidatas
 - Es **completamente inabordable** encontrar la ruta óptima usando métodos exhaustivos y completos, en un tiempo razonable

Metaheurística

- Cuando la técnica es heurística, pero lo suficientemente genérica como para poder aplicarla a problemas de distinto tipo, la denominamos **metaheurística**
- En este tema, veremos las siguientes técnicas **metaheurísticas para optimización**:
 - Búsqueda en escalada
 - Enfriamiento simulado
 - Algoritmos genéticos
 - PSO: *Particle Swarm Optimization*

Mejoras iterativas

- La mayoría de las técnicas que veremos en este tema se basan en la idea de *mejora iterativa*:
 - Empezar con un(os) candidato(s) inicial(es) cual(esquiera)
 - Mejorar su calidad paso a paso
- Búsqueda local
- “Mejorar”:
 - Usualmente con una componente aleatoria, probabilística
 - Heurística basadas en conocimiento previo sobre el problema
- Problema de los *óptimos locales*

Representación de un problema de optimización (para aplicar búsqueda local)

- Elección de una representación para los estados (estructura de datos)
- Función **F-VALORACIÓN (ESTADO)**
 - Función cuyo valor se trata de optimizar
 - Minimizar o maximizar
- Función **GENERA-ESTADO-INICIAL ()**
 - Si en el problema el estado inicial no está claramente definido, el estado inicial puede generarse de manera aleatoria, o usando alguna técnica heurística
- Función **GENERA-SUCESOR (ESTADO)**
 - Genera un estado sucesor a uno dado
 - Define la noción de “vecindad” para el problema concreto
 - Usualmente, existe cierta componente aleatoria y heurística en la generación del sucesor

Representación del problema del viajante

- Representación de los estados:
 - Cada estado será un posible circuito, representado por una lista con *todas* las ciudades, sin repetir, en un orden determinado
 - Ejemplo: (malaga cadiz cordoba almeria huelva granada jaen sevilla)

representa el viaje

Málaga → Cádiz → Córdoba → Almería → Huelva →
Granada → Jaen → Sevilla → Málaga

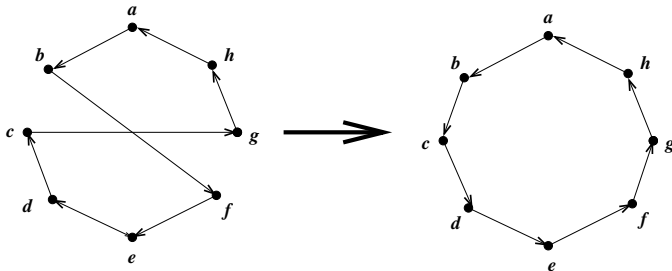
- Generación aleatoria del estado inicial
 - Asumiremos que la función **GENERA-ESTADO-INICIAL()** obtiene un circuito inicial aleatorio
- Función DE VALORACIÓN
 - La función **F-VALORACIÓN(ESTADO)** devuelve la distancia total del circuito

Representación del problema del viajante

- Generación de un sucesor con la función

GENERA-SUCESOR (ESTADO)

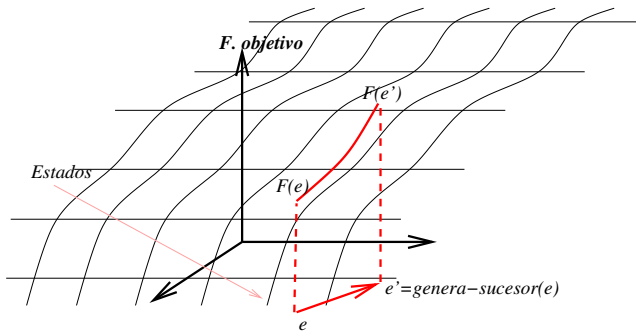
- Elección aleatoria de dos ciudades e inversión del camino entre ellas



- Heurística + aleatoriedad
 - Heurística: trata de reducir los “cruces”
 - Aleatoriedad: al elegir el par de ciudades

Búsqueda en escalada

- Idea de la búsqueda en escalada:
 - Aplicar una simple *mejora iterativa*
 - Guiado por la heurística y aleatoriedad de la función que genera sucesores
 - No se permite recuperarse de un camino erróneo
 - Puede verse como una escalada (ascenso o descenso)



Búsqueda en escalada

- 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**

Búsqueda en escalada: ejemplo

- Ejemplos

```
>>> 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)
```

- Ninguna de ellas es la solución óptima

Búsqueda en escalada

- Evidentemente, no se garantiza encontrar el óptimo
- Problemas:
 - Su eficacia depende en gran medida de la función
GENERA-SUCESOR
 - Óptimos locales, mesetas
- Una idea simple para intentar escapar de los óptimos locales:
 - Buscar aleatoriamente el inicio de la pendiente
 - Hacer escalada (o descenso) a partir de ahí
 - Iterar el proceso
 - Devolver el mejor estado conseguido en alguna de las iteraciones

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

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ 🔍 ↺

Escalada con reinicio aleatorio: ejemplo

- Problema del cuadrado de puntos:

```
>>> escalada_con_reinicio_aleatorio(Cuadrado_Puntos_BL(3),10000)
([(3, 3), (3, 2), (2, 3), (1, 3), (0, 3), (0, 2), (0, 1),
  (0, 0), (1, 0), (3, 1), (2, 0), (3, 0)],
 16.06449510224598)
>>> escalada_con_reinicio_aleatorio(Cuadrado_Puntos_BL(3),100000)
([(3, 0), (3, 1), (2, 0), (1, 0), (0, 0), (0, 1), (0, 2),
  (0, 3), (2, 3), (1, 3), (3, 3), (3, 2)],
 15.414213562373096)
```

- Soluciones no óptimas, debemos mejorar la técnica para escapar de los óptimos locales

Enfriamiento simulado

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

Enfriamiento simulado (inspiración físico-química)

- Enfriamiento de metales
 - Sistema estable: mínimo de energía
 - Dada una temperatura, el sistema necesita tiempo para estabilizarse (perder energía)
 - Es posible pasar momentáneamente por estados de mayor energía, con probabilidad dada por

$$p(\Delta E, T) = e^{-\frac{\Delta E}{k \cdot T}}$$

- Después de estabilizarse, se vuelve a bajar la temperatura, y el sistema se estabiliza nuevamente en un estado de menor energía
- Programa de enfriamiento
 - Al principio (T grande) mayor probabilidad de aceptación de soluciones candidatas (diversificación)
 - Al final (T pequeña), se aceptan pocas soluciones candidatas (intensificación)

Enfriamiento simulado

- Generación del estado inicial y estados vecinos
 - Aleatoria, heurística
- Aceptación probabilística
 - Probabilidad de aceptación de un estado que incrementa ΔF el valor de la función de valoración: $e^{-\frac{\Delta F}{T}}$
- Programa de enfriamiento, en nuestro caso:
 - Temperatura inicial
 - Variación de la temperatura: $T \leftarrow \alpha \cdot T$
 - Número fijo de iteraciones para cada T
- Criterio de parada
 - Valor suficientemente bueno de la función de valoración
 - Número máximo de iteraciones sin mejora
 - En nuestro caso, número fijo de iteraciones totales

Implementación de enfriamiento simulado (I)

**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

(continúa...)

Implementación de enfriamiento simulado (II)

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

Enfriamiento simulado: ejemplo

- Ejemplo (problema del viajante por Andalucía)

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

- Ejemplo (problema del cuadrado de puntos)

```
>>> enfriamiento_simulado(Cuadrado_Puntos_BL(3), 5, 0.95, 100, 100)
([(3, 0), (2, 0), (1, 0), (0, 0), (0, 1), (0, 2),
 (0, 3), (1, 3), (2, 3), (3, 3), (3, 2), (3, 1)],
 12.0)
>>> enfriamiento_simulado(Cuadrado_Puntos_BL(7), 5, 0.95, 100, 100)[1]
28.0
>>> enfriamiento_simulado(Cuadrado_Puntos_BL(10), 20, 0.95, 100, 100)[1]
40.0
>>> enfriamiento_simulado(Cuadrado_Puntos_BL(15), 35, 0.95, 100, 100)[1]
113.3125429928352
>>> enfriamiento_simulado(Cuadrado_Puntos_BL(15), 35, 0.95, 500, 500)[1]
60.0
```

Algoritmos genéticos: evolución natural

- Optimización inspirada en los procesos evolutivos de la naturaleza:
 - La evolución ocurre en los cromosomas de los individuos
 - Las “buenas estructuras” sobreviven con más probabilidad que las demás
 - El nuevo material genético se obtiene mediante cruces y mutaciones
- Algoritmos genéticos:
 - Aplicación de estas ideas en la búsqueda de soluciones óptimas
 - No existe un único algoritmo genético
 - Es una denominación para este tipo de algoritmos evolutivos

Algoritmos genéticos: representación del problema

- Problemas de optimización: un ejemplo simple
 - Ejemplo: encontrar el mínimo de la función $f(\mathbf{x}) = \mathbf{x}^2$ en $[0, 2^{10}) \cap \mathbf{N}$
- Variables ***GENES*** y ***LONGITUD-INDIVIDUOS***
 - Ejemplo en la función cuadrado: $[0, 1]$ y 10, resp.
- Función **DECODIFICA**(\mathbf{x}), obtiene el *fenotipo*
 - En la función cuadrado: un cromosoma puede verse como un número binario de 10 dígitos (en orden inverso). El fenotipo de un cromosoma es dicho número (en notación decimal)
 - Ejemplo: (0 1 1 0 0 1 0 0 0 0) es un cromosoma que representa al 38
- Función **FITNESS**(\mathbf{x}), valor de de la función a optimizar (actuando sobre el genotipo)
 - Ejemplo en la función cuadrado: la función que recibiendo la representación binaria de un número, devuelve su cuadrado

Algoritmos genéticos: mecanismo evolutivo

- Evolución de las sucesivas generaciones
 - Operadores de variación:
 - Cruces: combinación de estructuras (*padres* que dan lugar a *hijos*)
 - Mutaciones: cambio de algunos genes
 - Mecanismos de selección
 - Sirven tanto para decidir qué cromosomas se van a usar para combinación, como para decidir quiénes pasan a la generación siguiente
 - En general, debe ser un método de selección que favorezca a los individuos con mejor valoración, pero que no impida la *diversidad*
- En general, es bastante común que se maneje aleatoriedad

Esquema general de un algoritmo genético

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

Ejemplo de ejecución (minimizando la función cuadrado)

```
>>> algoritmo_genetico_v2_con_salida(cuad_gen, 20, 10, 0.75, 0.6, 0.1)
```

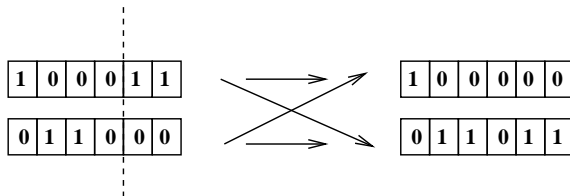
```

Generacion: 1. Media: 361954.9, Mejor: (0 1 1 0 0 1 0 0 0 0), Valor: 1444
Generacion: 2. Media: 79730.6, Mejor: (0 1 1 0 0 1 0 0 0 0), Valor: 1444
Generacion: 3. Media: 22278.6, Mejor: (0 1 1 0 0 1 0 0 0 0), Valor: 1444
Generacion: 4. Media: 3537.7, Mejor: (1 1 1 1 0 0 0 0 0 0), Valor: 225
Generacion: 5. Media: 1597.3, Mejor: (0 0 1 1 0 0 0 0 0 0), Valor: 144
Generacion: 6. Media: 912.8, Mejor: (0 1 0 0 0 0 0 0 0 0), Valor: 4
Generacion: 7. Media: 345.3, Mejor: (0 1 0 0 0 0 0 0 0 0), Valor: 4
Generacion: 8. Media: 60.7, Mejor: (0 1 0 0 0 0 0 0 0 0), Valor: 4
Generacion: 9. Media: 14.0, Mejor: (0 1 0 0 0 0 0 0 0 0), Valor: 4
Generacion: 10. Media: 4.5, Mejor: (0 1 0 0 0 0 0 0 0 0), Valor: 4
Generacion: 11. Media: 3.7, Mejor: (1 0 0 0 0 0 0 0 0 0), Valor: 1
Generacion: 12. Media: 3.4, Mejor: (1 0 0 0 0 0 0 0 0 0), Valor: 1
Generacion: 13. Media: 2.4, Mejor: (0 0 0 0 0 0 0 0 0 0), Valor: 0
....

```

Combinación de individuos

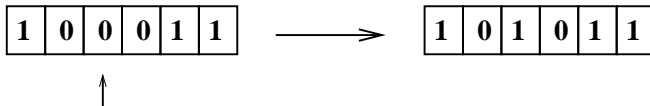
- Operadores que combinan la información de los *padres* para obtener nuevos *hijos*
- Cruce en un punto:



- Aleatoriedad: al elegir el punto de cruce
- Otras posibilidades:
 - Cruce multipunto (varios segmentos de intercambio)
 - Cruce uniforme (para cada posición del hijo, *sorteamos* de quién hereda)
 - Cruces específicos de la representación (p.ej. permutaciones)

Mutación de individuos

- Mutaciones:



- Aleatoriedad:

- Con una determinada probabilidad (usualmente baja) cambiar algunos genes
- Si se cambia, elegir aleatoriamente a qué gen se cambia

- Variantes:

- Específicas de la representación (p.ej. permutaciones)

Permutaciones

- En caso en que el cromosoma sea una permutación de genes, es necesario tener operadores específicos de mutación y cruce
- Mutación por intercambio:



- Mutación por inserción:



- Mutación por mezcla:



Permutaciones

Cruce basado en orden

- Elige dos puntos de corte aleatoriamente del primer padre y copia el segmento entre ellos en el primer hijo.
- A partir del segundo punto de corte en el segundo padre, copia los genes no usados en el primer hijo en el mismo orden en que aparecen en el segundo padre, volviendo al principio de la lista si fuera necesario.
- Crea el segundo hijo de manera análoga, intercambiando el rol de los padres.

Permutaciones

Cruce basado en orden (Paso 1):

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



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

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

Cruce basado en orden (Paso 2):

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



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

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

Permutaciones

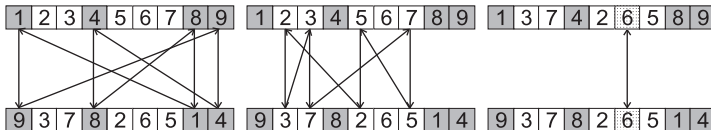
Cruce basado en ciclos: Dividimos la permutación en ciclos y alternamos los ciclos de cada padre.

Para construir ciclos:

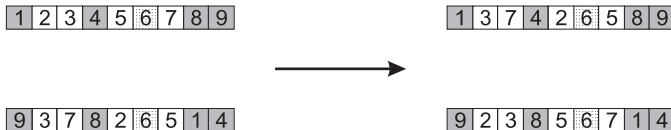
- Tomamos la primera posición *nueva* en el padre P_1 .
- Buscamos el gen en la misma posición de P_2 .
- Vamos a la posición con el mismo gen en P_1 .
- Añadimos este gen al ciclo.
- Repetimos los pasos del 2 al 4 hasta que lleguemos al primer gen de P_1 .

Permutaciones

Cruce basado en ciclos (Identificación de ciclos):



Cruce basado en ciclos (Construcción de hijos):



Mecanismos de selección

- Un algoritmo genético debe tener un método para seleccionar individuos de una población:
 - Para obtener aquellos individuos que van a ser usados como padres
 - A veces, también para decidir qué hijos pasan a la siguiente generación
- La selección debe:
 - Favorecer a los mejores (según su valoración)
 - Permitir la diversidad
 - Usualmente tiene una componente aleatoria
- Ejemplos de selección:
 - Proporcional a su valoración
 - Torneo
 - Élite + aleatoriedad

Selección proporcional a la valoración

- Idea:
 - Seleccionar aleatoriamente, pero de manera que cada individuo tenga una probabilidad de ser seleccionado proporcional a su valoración respecto de las valoraciones del total de la población
 - Los mejores individuos se seleccionarán más frecuentemente
- La probabilidad de que cada individuo i sea seleccionado es

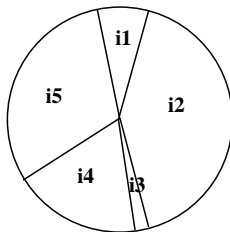
$$P(i) = \frac{F(i)}{\sum_{j=1}^n F(j)}$$

- Importante: con este método de selección sólo podemos resolver problemas de maximización
 - Si es de minimización habría que transformar la función de *fitness*
- Variante: selección por *ranking*
 - La probabilidad asignada es proporcional a su posición en la población (ordenada por *fitness*)

Selección probabilística

- Implementación de sorteos con probabilidad: ruleta
 - Calcular la suma total acumulada de los valores de la función de *fitness* de todos los miembros de la población
 - Generar un número aleatorio x entre 0 y la suma total anterior
 - Recorrer la población, nuevamente acumulando los valores de la función *fitness* y seleccionando el primer cromosoma cuya suma acumulada sea mayor que x

Ejemplo del método de la ruleta



- Población de 5 individuos $\{i1, i2, i3, i4, i5\}$ con valores $F(i1) = 2$, $F(i2) = 7$, $F(i3) = 1$, $F(i4) = 4$, $F(i5) = 6$.
- Calculamos las sumas acumuladas:
 - $Ac(i1) = 2$,
 - $Ac(i2) = 2 + 7 = 9$,
 - $Ac(i3) = 2 + 7 + 1 = 10$,
 - $Ac(i4) = 2 + 7 + 1 + 4 = 14$,
 - $Ac(i5) = 2 + 7 + 1 + 4 + 6 = 20$.
- Para seleccionar cuatro individuos tomamos cuatro valores aleatorios entre 0 y 20: 7 13 17 5

Selección por torneo y élite

- Selección por torneo:
 - Para cada selección, extraer aleatoriamente k individuos y seleccionar el mejor
 - Ventajas: no depende de la magnitud de la función de la valoración y se puede aplicar tanto a minimización como a maximización
 - Cuanto mayor el k usado, mayor es la *presión evolutiva*
- Selección elitista:
 - Escoger directamente un porcentaje de los mejores
 - Para introducir diversidad, el resto, seleccionarlos aleatoriamente de entre el resto

Otras componentes de un algoritmo genético

- Población inicial
 - Usualmente se crean ***N*** individuos de manera completamente aleatoria
- Terminación del algoritmo:
 - Hasta completar un número dado de generaciones
 - Cuando se hayan completado un número dado de generaciones sin mejorar
 - Hasta un valor prefijado de la función de valoración
- Diversos parámetros:
 - Tamaño de la población
 - Proporción de padres
 - Probabilidades de cruce y/o mutación

Ejemplo de algoritmo genético

```

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)

```

- Selección mediante torneo
- Parámetros del algoritmo: tamaño de la población, número de generaciones, proporción de padres (r), probabilidad de mutación

Otro ejemplo de algoritmo genético

```

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)

```

- Selección combinada élite y aleatoriedad
- Para la siguiente generación, se toman los mejores de entre los hijos y los individuos originales
- Parámetros del algoritmo: tamaño de la población, número de generaciones, proporción de padres, proporción de mejores entre los padres, probabilidad de mutación

Ejemplos: problema del cuadrado mágico

- Colocar en un cuadrado $n \times n$ los números naturales de 1 a n^2 , de tal manera que las filas, las columnas y las diagonales principales sumen los mismo
- Casos $n = 3$ y $n = 4$:

4	3	8
9	5	1
2	7	6

7	14	9	4
16	2	5	11
1	15	12	6
10	3	8	13

- Soluciones representadas como permutaciones de números entre 1 y n^2
- Genes y longitud de los individuos:
 - *GENES* = (1 2 3 ... n^2)
 - *LONGITUD-INDIVIDUOS* = n^2

Ejemplos: problema de la mochila

- Problema:
 - Dados n objetos de pesos p_i y valor v_i ($i = 1, \dots, n$), seleccionar cuáles se meten en una mochila que soporta un peso P máximo, de manera que se maximice el valor total de los objetos introducidos
- Genes y longitud de los individuos en el problema de la mochila
 - ***GENES*** = [0, 1]
 - ***LONGITUD-INDIVIDUOS*** = n

Ejemplos: problema de la mochila

- Función de decodificación, **DECODIFICA (X)** :
 - 1 ó 0 representan, respectivamente, si el objeto se introduce o no en la mochila
 - Tomados de izquierda a derecha, a partir del primero que no cabe, se consideran todos fuera de la mochila, *independientemente del gen en su posición*
 - De esta manera, todos los individuos representan candidatos válidos
- **FITNESS (X)** :
 - Suma de valores de los objetos, que según la representación dada por el **DECODIFICA** anterior, están dentro de la mochila
 - Nótese que en este caso no es necesaria ninguna penalización

Algoritmos Genéticos

Aplicaciones

Journal List > Croat Med J > v.60(2); 2019 Apr > PMC6509630

<i>Free full text at www.cmj.hr</i>				
Info for Authors	Submit	Subscribe	About	

[Croat Med J](#). 2019 Apr; 60(2): 177–180.

doi: [10.3325/cmj.2019.60.177](https://doi.org/10.3325/cmj.2019.60.177)

PMCID: PMC6509630

PMID: [31044592](https://pubmed.ncbi.nlm.nih.gov/31044592/)

Genetics without genes: application of genetic algorithms in medicine

[Branimir K. Hackenberger](#)

Algoritmos Genéticos

Aplicaciones

npj | Computational Materials

Article | [Open Access](#) | Published: 10 April 2019

Genetic algorithms for computational materials discovery accelerated by machine learning

Paul C. Jennings, Steen Lysgaard, Jens Strabo Hummelshøj, Tejs Vegge  & Thomas Bligaard

Optimización de enjambre de partículas

Particle Swarm Optimization (PSO)

- Inspirado en el vuelo de una bandada de pájaros.
- Técnica relacionada con el vuelo de murciélagos en *Batman Returns*¹ o la estampida de bisontes en *El Rey León*.
- Se enmarca en el modelo conocido como *Inteligencia de Enjambre (Swarm Intelligence)*, que estudia el comportamiento colectivo de sistemas descentralizados auto-organizativos.

¹I.V. Kerlow. The Art of 3D: Computer Animation and Effects. Wiley John & Son, 2009. p23.

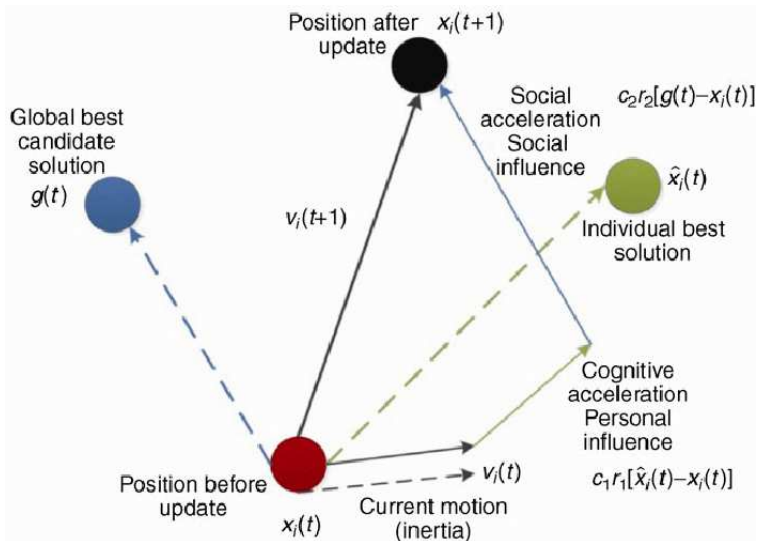
Optimización de enjambre de partículas

Un sistema PSO consta de n partículas que ocupan una posición en \mathbb{R}^m y que llevan asociado un vector de velocidad, junto con una función fitness que asocia un valor a cada posición. El objetivo es encontrar la posición donde la función fitness toma el valor mínimo (o máximo).

- La posición y velocidad inicial de las partículas se establece aleatoriamente.
- Todas las partículas actualizan su posición en paralelo.
- Todas las partículas 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 $t + 1$

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

Optimización de enjambre de partículas



Optimización de enjambre de partículas

Ejemplo

- Optimización en \mathbb{R}^2
- Función fitness $f(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - 0,9)^2 + (\mathbf{y} - 0,9)^2$, con mínimo en $(0,9, 0,9)$.
- $W = 0,5$, $c_1 = 0,5$, $c_2 = 0,5$
- 50 partículas. Las posiciones iniciales toman valores en $[0, 1]^2$ y las velocidades iniciales en $[-1, 1]^2$.
- Evolución de la mejor partícula:

Paso 0 – Pos: (0.7568, 0.9482) – **F:** 0.0228

Paso 1 – Pos: (0.8968, 0.8576) – **F:** 0.0018

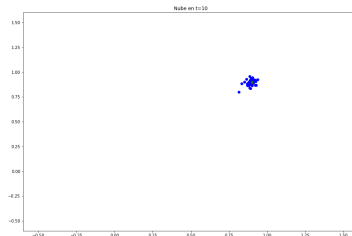
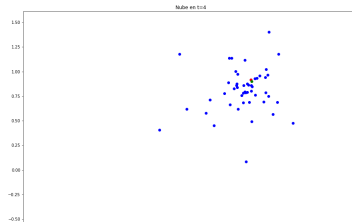
...

Paso 4 – Pos: (0.8956, 0.9155) – **F:** 0.0002

...

Paso 10 – Pos: (0.9003, 0.8995) – **F:** 3.10 e-07

Pos. óptima en verde - Mejor pos. hasta t en rojo



Optimización de enjambre de partículas

Aplicaciones

Mathematical Problems in Engineering
Volume 2019, Article ID 8164609, 10 pages

<https://doi.org/10.1155/2019/8164609>

Research Article

Application of Improved Particle Swarm Optimization in Vehicle Crashworthiness

Dawei Gao , Xiangyang Li, and Haifeng Chen

School of Mechanical Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China

Correspondence should be addressed to Dawei Gao;
gddwww1999@163.com

Optimización de enjambre de partículas






Aplicaciones

ORIGINAL RESEARCH ARTICLE

Front. Chem., 12 July 2019 | <https://doi.org/10.3389/fchem.2019.00485>



Modified Particle Swarm Optimization Algorithms for the Generation of Stable Structures of Carbon Clusters, C_n ($n = 3-6, 10$)

 Gourhari Jana¹,  Arka Mitra²,  Sudip Pan³,  Shamik Sural^{4*} and  Pratim K. Chattaraj^{1,5*}

Bibliografía

- Floreano, D. y Mattiussi, C. *Bio-Inspired Artificial Intelligence* (The MIT Press, 2008)
 - Cap. 7: “Collective Systems”.
- Eibe, A.E. y Smith, J.E. *Introduction to Evolutionary Computing* (Springer, 2007).
 - Cap. 3: “Genetic Algorithms”.
- Russell, S. y Norvig, P. *Artificial Intelligence (A Modern Approach) 3rd edition* (Prentice–Hall, 2010).
 - Cap. 4 “Beyond classical search”.
- Mitchell, T.M. *Machine Learning* (McGraw-Hill, 1997)
 - Cap. 9: “Genetic Algorithms”
- Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs* (Springer, 1999).
 - Cap. 2 “GAs: How Do They Work?”.