7. ALGORITMOS INSPIRADOS EN LA NATURALEZA

IA 3.2 - Programación III

1° C - 2023

Lic. Mauro Lucci

La naturaleza como fuente de inspiración

- Observar cómo la naturaleza resuelve ciertos problemas y simularlo para un problema propio.
- Algoritmos inspirados en fenómenos biológicos, leyes físicas, procesos químicos, etc.
- La familia de algoritmos inspirados en la naturaleza es muy amplia.
- Han demostrado ser muy efectivos en problemas de optimización.
- Al día de hoy, sigue siendo un área muy productiva en el diseño de algoritmos.

Gran variedad de algoritmos de búsqueda inspirados en la naturaleza.

Swarm intelligence based algorithms	
Algorithm	Author
Accelerated PSO	Yang et al.
Ant colony optimization	Dorigo
Artificial bee colony	Karaboga and Basturk
Bacterial foraging	Passino
Bacterial-GA Foraging	Chen et al.
Bat algorithm	Yang
Bee colony optimization	Teodorović and Dell'Orco
Bee system	Lucic and Teodorovic
BeeHive	Wedde et al.
Wolf search	Tang et al.
Bees algorithms	Pham et al.
Bees swarm optimization	Drias et al.
Bumblebees	Comellas and Martinez
Cat swarm	Chu et al.
Consultant-guided search	Iordache
Cuckoo search	Yang and Deb
Eagle strategy	Yang and Deb
Fast bacterial swarming algorithm	Chu et al.
Firefly algorithm	Yang
Fish swarm/school	Li et al.
Good lattice swarm optimization	Su et al.
Glowworm swarm optimization	Krishnanand and Ghose
Hierarchical swarm model	Chen et al.
Krill Herd	Gandomi and Alavi
Monkey search	Mucherino and Seref
Particle swarm algorithm	Kennedy and Eberhart
Virtual ant algorithm	Yang
Virtual bees	Yang
Weightless Swarm Algorithm	Ting et al.
Other algorithms	
Anarchic society optimization	Shayeghi and Dadashpour
Artificial cooperative search	Civicioglu
Backtracking optimization search	Civicioglu
Differential search algorithm	Civicioglu
Grammatical evolution	Ryan et al.
Imperialist competitive algorithm	Atashpaz-Gargari and Lucas
League championship algorithm	Kashan
Social emotional optimization	Xu et al.

Bio-inspired (not SI-based) algorithms	
Algorithm	Author
Atmosphere clouds model	Yan and Hao
Biogeography-based optimization	Simon
Brain Storm Optimization	Shi
Differential evolution	Storn and Price
Dolphin echolocation	Kaveh and Farhoudi
Japanese tree frogs calling	Hernández and Blum
Eco-inspired evolutionary algorithm	Parpinelli and Lopes
Egyptian Vulture	Sur et al.
Fish-school Search	Lima et al.
Flower pollination algorithm	Yang
Gene expression	Ferreira
Great salmon run	Mozaffari
Group search optimizer	He et al.
Human-Inspired Algorithm	Zhang et al.
Invasive weed optimization	Mehrabian and Lucas
Marriage in honey bees	Abbass
OptBees	Maia et al.
Paddy Field Algorithm	Premaratne et al.
Roach infestation algorithm	Havens
Queen-bee evolution	Jung
Shuffled frog leaping algorithm	Eusuff and Lansey
Termite colony optimization	Hedayatzadeh et al.
Physics and Chemistry based algorithms	
Big bang-big Crunch	Zandi et al.
Black hole	Hatamlou
Central force optimization	Formato
Charged system search	Kaveh and Talatahari
Electro-magnetism optimization	Cuevas et al.
Galaxy-based search algorithm	Shah-Hosseini
Gravitational search	Rashedi et al.
Harmony search	Geem et al.
Intelligent water drop	Shah-Hosseini
River formation dynamics	Rabanal et al.
Self-propelled particles	Vicsek
Simulated annealing	Kirkpatrick et al.
Stochastic difusion search	Bishop
Spiral optimization	Tamura and Yasuda
Water cycle algorithm	Eskandar et al.

Recocido Simulado

Simulated Annealing (SA)

El **recocido** es un tratamiento térmico que permite alterar la estructura molecular de un material para mejorar sus propiedades mecánicas (ductilidad, dureza, tensiones internas).

1. **Calentar** un material por encima de su temperatura de recristalización.
2. **Mantener** la temperatura por un período de tiempo
3. **Enfriar** progresivamente hasta alcanzar la temperatura ambiente.

Recocido simulado

El **recocido simulado** es un algoritmo de búsqueda local con un mecanismo para escapar de máximos locales inspirado en la etapa de enfriamiento del proceso de recocido.

- Permite moverse a un vecino que empeore la función objetivo con una cierta **probabilidad**.
- Esta probabilidad decrece a medida que el sistema se **enfría**.
- Es decir, los “malos” movimientos son más probables al comienzo y se vuelven menos probables a medida que la temperatura baja.

Recocido simulado

Idea.

1. Comienza en un estado inicial con una temperatura inicial.
2. En cada iteración:
    - La temperatura **disminuye.**
    - Se elige al **azar** un estado sucesor *s'* del estado actual *s*.
    - Se acepta *s'* como nuevo estado actual con una cierta **probabilidad**.
3. El algoritmo se detiene cuando la temperatura alcanza un valor final.

Esquema de recocido

El **esquema de recocido** es una función que determina la temperatura en función del tiempo (número de iteraciones).

Típicamente, se da en función de una **temperatura inicial** t_0 y de un **factor de enfriamiento** $c \in (0,1)$.

Es decir, la temperatura t inicialmente es

$$t = t_0$$

y en cada iteración se enfría

$$t = c.t$$

Probabilidad de aceptación

La función de **probabilidad de aceptación** $P(s,s',t)$ determina la probabilidad de aceptar el estado sucesor s' de s como nuevo estado actual cuando la temperatura es t.

- Si $h(s) < h(s')$,

$$P(s,s',t) = 1.$$

- Sino,

$$P(s,s',t) > 0,$$

  y debe tender a 0 cuando t disminuye.

Algoritmo de recocido simulado

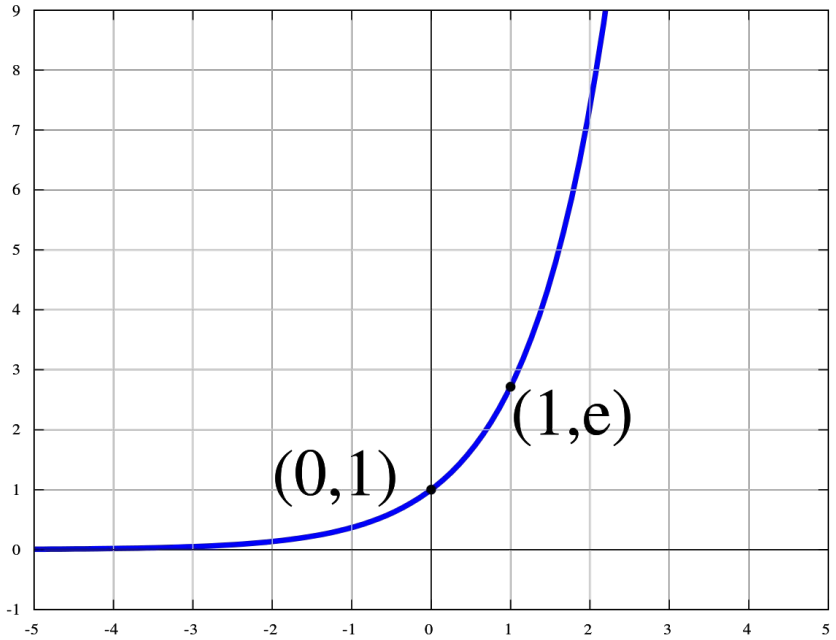
```
1 function RECOCIDO-SIMULADO(problema, esquema, P) return estado
2    actual ← nodo con el estado inicial del problema
3    for i = 1 to ∞
4         t ← esquema(i)
5         if t = 0 then return actual
6         vecino ← nodo con un estado sucesor de actual elegido al azar
7         ΔE ← vecino.valor - actual.valor
8         if ΔE > 0 then
9              actual ← vecino
10        else if random(0,1) < P(actual.estado, vecino.estado, t)
11             actual ← vecino
```

📝 Ejemplo

Típicamente, la función de probabilidad de aceptación para sucesores que empeoren el estado actual es:

$$p(s, s', t) = e^{\frac{h(s') - h(s)}{t}}$$

Así, la probabilidad es exponencialmente más baja cuanto peor es el estado sucesor y menor la temperatura.



Gráfica de f(x) = e^x.

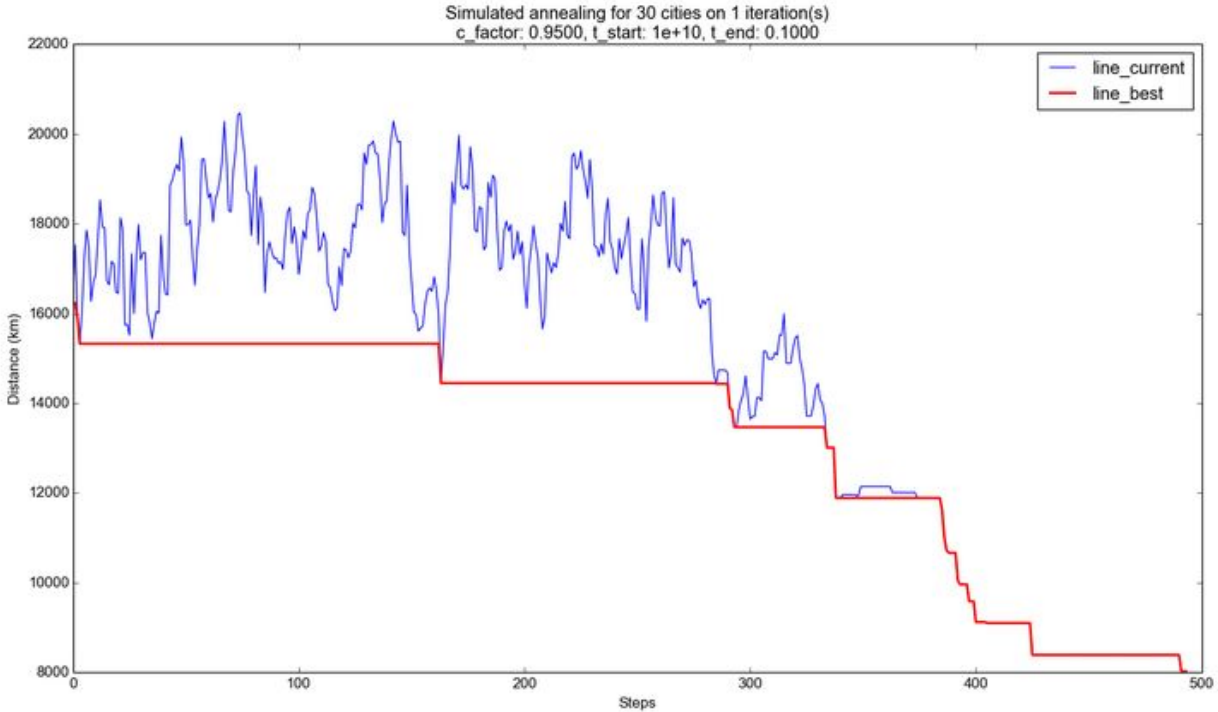
📝 Ejemplo

La temperatura inicial t_0 se suele determinar de manera que la probabilidad de aceptar un estado que sea peor en un *w%* al estado inicial s_0 sea igual a 0,5.

$$e^{\frac{(1-w).h(s_0)-h(s_0)}{t_0}} = 0,5 \quad \Longrightarrow \quad t_0 = \frac{-w.h(s_0)}{\log(0,5)}$$

El valor más adecuado para *w* y *c* (factor de enfriamiento) se determina por **experimentación**.

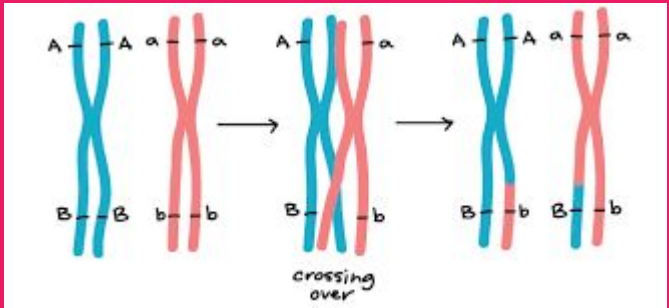
Ejemplo



Número de iteraciones vs. valor objetivo del estado actual (azul) y del mejor estado (rojo) para un TSP de 30 ciudades resuelto con SA.

Algoritmo Genético

Genetic Algorithm (GA)



Los **algoritmos genéticos** se inspiran en la teoría evolutiva de la **selección natural**.

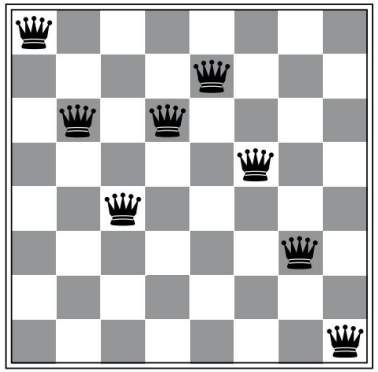
- Se inicia con una población de individuos.
- En cada iteración, se **seleccionan** los mejores individuos de la generación actual.
- Los genomas de los individuos seleccionados se **cruzan** y se **mutan** para formar la siguiente generación.
- Típicamente, el algoritmo termina cuando se alcanza cierto número de generaciones.

1. Población inicial

Un GA comienza con una **población inicial** formada por un conjunto de *k* estados (o **individuos**) generados aleatoriamente.

El genoma de cada individuo se representa con una string sobre un alfabeto finito.

📝 **Ejemplo (8-Reinas).**



1 3 5 3 2 4 6 8

El i-ésimo dígito es la fila en la que se encuentra la reina de la columna i.

2. Función de fitness

Cada individuo se califica con una **función de fitness** (o función objetivo).

La función de fitness debe devolver valores más altos para los mejores individuos.

📝 **Ejemplo (8-Reinas).**

Una función de fitness puede ser el número de pares de reina que no se atacan.

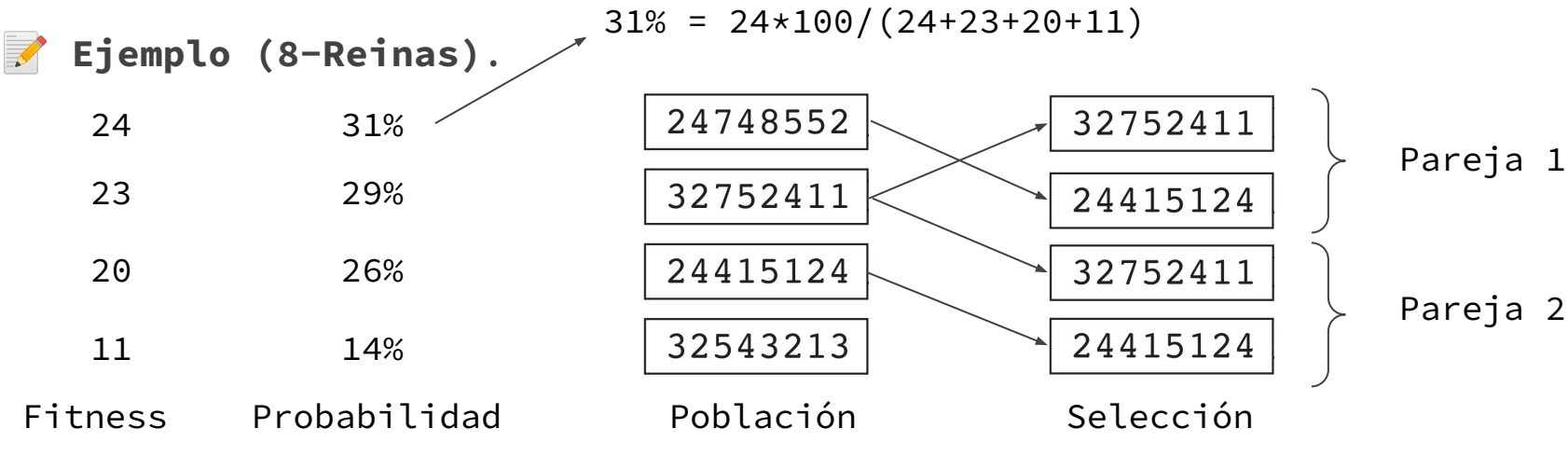
Los estados objetivo tienen un valor de 28 (formas de tomar 2 elementos de un conjunto de 8).

3. Selección

La **selección** consiste en elegir al azar pares de individuos de la generación actual para reproducirlos.

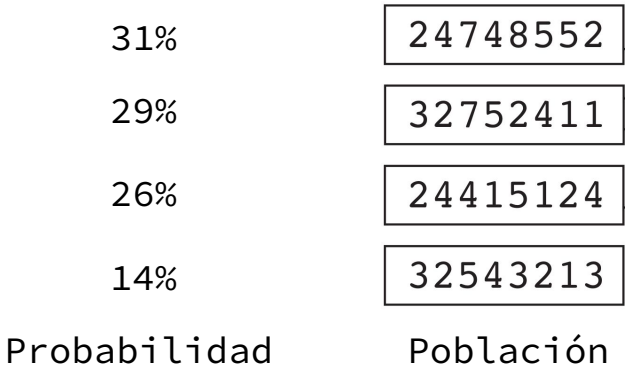
Usualmente, la probabilidad de que un individuo sea elegido es directamente proporcional a su valor de fitness.

📝 **Ejemplo (8-Reinas).**



3. Selección

¿Cómo elegir a un individuo al azar con probabilidades diferenciadas?



Generemos un número entero aleatorio r entre 1 y 100.

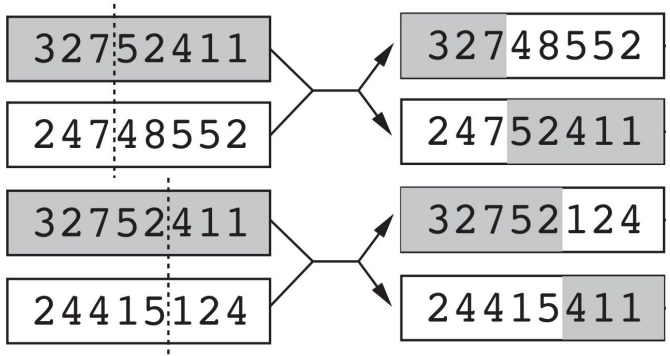
$1 \leq r \leq 31$, individuo 1.
$32 \leq r \leq 60$, individuo 2.
$61 \leq r \leq 86$, individuo 3.
$87 \leq r \leq 100$, individuo 4.

4. Cruzamiento

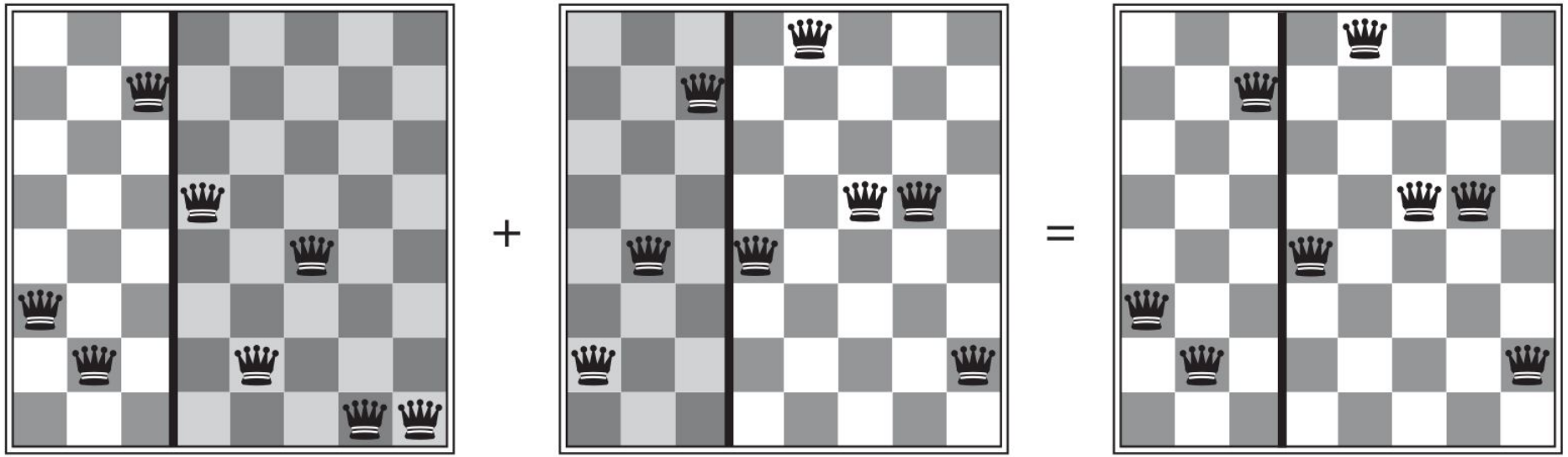
El **cruzamiento** consiste en generar nuevos individuos a partir de la información genética de sus padres.

Típicamente, se elige una posición *i* aleatoria de la string y se combinan los primeros *i* caracteres del primer individuo con los caracteres restantes del segundo individuo.

📝 **Ejemplo (8-Reinas).**



4. Cruzamiento



5. Mutación

La **mutación** consiste en realizar con una probabilidad baja un cambio aleatorio en la información genética del nuevo individuo.

Típicamente, se elige una posición *i* aleatoria de la string y se cambia su valor.

📝 **Ejemplo (8-Reinas).**

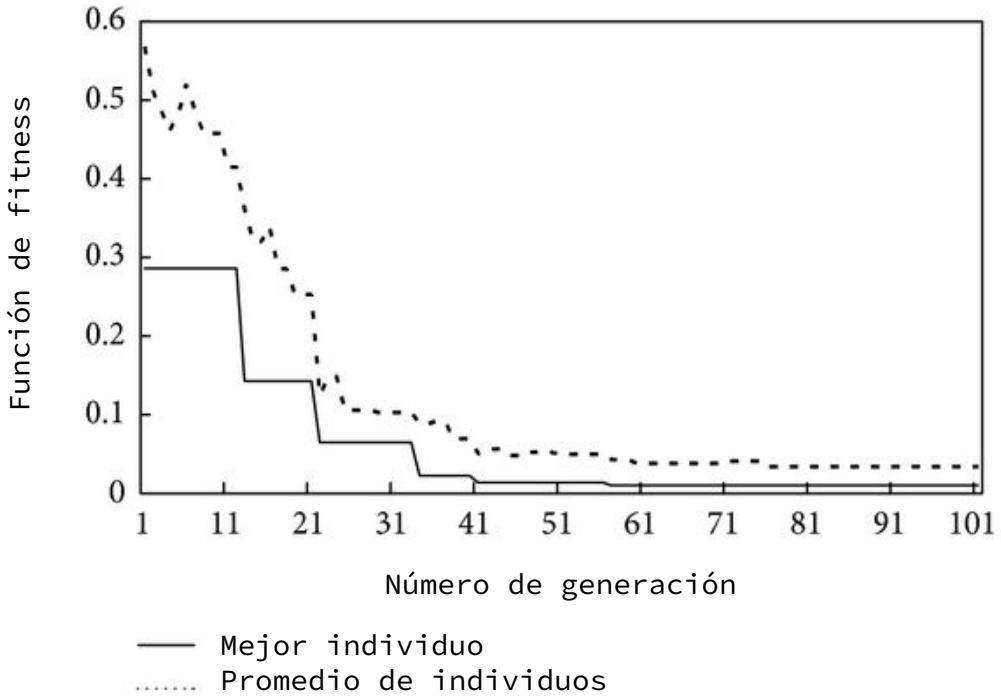
Esta mutación representa mover a la reina de la columna 6 de la fila 5 a la 1.

Ejemplo (8-Reinas)

Algoritmo genético

```
1 function ALGORITMO-GENÉTICO(población, fitness) return individuo
2   while no se cumple el criterio de parada do
3        población' ← {}
4        for i = 1 to len(población)
5             x ← SELECCIÓN(población, fitness)
6             y ← SELECCIÓN(población, fitness)
7             hijo ← CRUZAR(x, y)
8             if probabilidad baja then hijo ← MUTAR(hijo)
9             población’ ← población’ U {hijo}
10       población ← población’
11  return argmax{fitness(individuo): individuo ∈ población}
```

Evolución de las generaciones



Inicialmente, la población es bastante diversa, luego el cruzamiento produce individuos muy diferentes a sus padres.

Hacia las últimas generaciones, los individuos son muy similares, luego el cruzamiento produce individuos muy parecidos a sus padres.

📝 Ejercicio

Dar las componentes necesarias para un algoritmo genético que resuelva el TSP.

📝 Respuesta

- **Representación de camino**. Con strings sobre el alfabeto {1,…,n} según el orden en que se visitan las ciudad.

  El tour 2 → 3 → 1 → 5 → 4 → 2 sería la string 2 3 1 5 4

- **Función de fitness**. Costo del tour.

- **Cruzamiento**. El cruzamiento clásico puede romper el tour.

  1 3 | 4 2 5 **1** 3 5 **1** 2

  4 3 | 5 1 2 **4** 3 **4** 2 5

  Podemos arreglarlo cambiando ciudades repetidas por ciudades no visitadas.

  1 3 5 **1** 2 → 1 3 5 **4** 2 4 3 **4** 2 5 → 4 3 **1** 2 5

- **Mutación**. La mutación clásica rompe el tour. En su lugar, podemos aplicar un movimiento de 2-opt entre dos arcos aleatorios del tour.

📝 Respuesta

Representación ordinal.

Permite usar el cruzamiento y la mutación clásicos.

Dado el orden

Orden = 1 2 3 4 5

y el tour

Tour = 3 5 2 4 1

¿Cómo representamos Tour?

Tour	Orden	Repr.
3	1 2 3 4 5	3

El 1er elemento de Tour es 3 y está 3ro en Orden, entonces lo representamos con un 3.

Sacamos a 3 de Orden y seguimos…

📝 Respuesta

Representación ordinal.

Permite usar el cruzamiento y la mutación clásicos.

Dado el orden

Orden = 1 2 3 4 5

y el tour

Tour = 3 5 2 4 1

¿Cómo representamos Tour?

Tour	Orden	Repr.
3	1 2 3 4 5	3
5	1 2 4 5	4

El 2do elemento de Tour es 5 y está 4to en el nuevo Orden, entonces lo representamos con un 4.

Sacamos a 5 de Orden y seguimos…

📝 Respuesta

Representación ordinal.

Permite usar el cruzamiento y la mutación clásicos.

Dado el orden

Orden = 1 2 3 4 5

y el tour

Tour = 3 5 2 4 1

¿Cómo representamos Tour?

Tour	Orden	Repr.
3	1 2 3 4 5	3
5	1 2 4 5	4
2	1 2 4	2
4	1 4	2
1	1	1

Siguiendo este procedimiento, llegamos a la representación 3 4 2 2 1.

📝 Respuesta

Otras representaciones y operadores de cruzamiento y mutación para el TSP que existen en la literatura.

Representation	Operators	Authors
Binary	Classical + repair operator	Lidd (1991)
Path	Partially-mapped crossover	Goldberg and Lingle (1985)
	Order-crossover	Davis (1985)
	Order based crossover	Syswerda (1991)
	Position based crossover	Syswerda (1991)
	Heuristic crossover	Grefenstette (1987b)
	Edge recombination crossover	Whitley et al. (1989)
	Sorted match crossover	Brady (1985)
	Maximal preservative crossover	Mühlenbein et al. (1988)
	Voting recombination crossover	Mühlenbein (1989)
	Alternating-positions crossover	Larrañaga et al. (1996a)
	Displacement mutation	Michalewizc (1992)
	Exchange mutation	Banzhaf (1990)
	Insertion mutation	Fogel (1988)
	Simple inversion mutation	Holland (1975)
	Inversion mutation	Fogel (1990)
	Scramble mutation	Syswerda (1991)
Adjacency	Alternating edge crossover	Grefenstette et al. (1985)
	Subtour chunks crossover	Grefenstette et al. (1985)
	Heuristic crossover 1	Grefenstette et al. (1985)
	Heuristic crossover 2	Jog et al. (1989)
	Heuristic crossover 3	Suh and Van Gucht (1987)
Ordinal	Classical operators	Grefenstette et al. (1985)
Matrix	Intersection crossover operator	Fox and McMahon (1987)
	Union crossover operator	Fox and McMahon (1987)
	Repair operators	Seniw (1981)
	Repair operators	Homaifar and Guan (1991)
	Heuristic inversion mutation	Homaifar and Guan (1991)

Optimización por Colonia de Hormigas

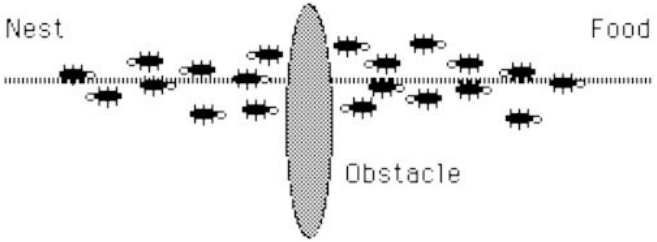
Ant Colony Optimization (ACO)

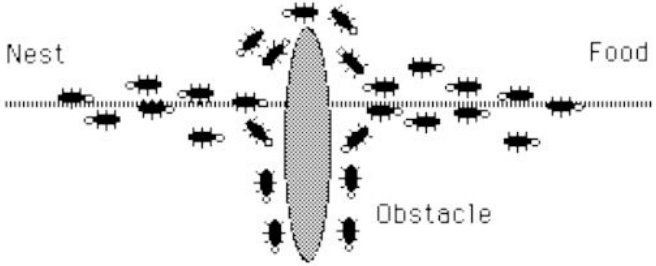
La **optimización por colonia de hormigas** se inspira en el rastro de **feromonas** que usan las hormigas como medio de comunicación.

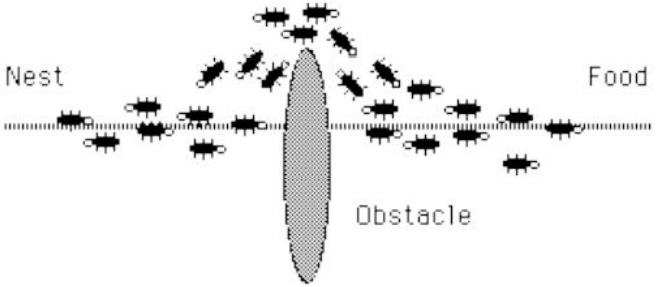
1. En cada iteración, una población de hormigas artificiales construye soluciones al problema guiadas por un rastro de feromonas.

2. Antes de pasar a la siguiente iteración, se ajusta la cantidad de feromonas por **depósito** y **evaporación**, para reflejar la experiencia acumulada durante la última búsqueda.

Analogía biológica







Algoritmo de ACO

```
1 function ACO(...) return solución
2    while no se cumple el criterio de parada do
3         cada hormiga construye una solución
4         actualizar el rastro de feromonas
5    return mejor solución encontrada
```

1. Construcción de soluciones

Una población de *m* hormigas construye soluciones de forma **incremental** y **estocástica**.

- Cada hormiga empieza con una solución parcial inicial.

  📝 **Ejemplo (TSP).** Inicia en una ciudad aleatoria.

- En cada paso de construcción, la hormiga elige un componente de solución al azar y aumenta su solución parcial.

  📝 **Ejemplo (TSP).** Viaja a una ciudad no visitada.

- La probabilidad de elegir un componente de solución depende del rastro de feromonas y de la función de evaluación.

📝 Ejemplo – TSP

Cada hormiga empieza en una ciudad aleatoria y, en cada paso de construcción, la hormiga viaja a una ciudad no visitada.

Estando en i, la probabilidad de moverse a j es directamente proporcional a:

$$\frac{\tau(i,j)}{dist(i,j)}$$

donde $\tau(i,j)$ es la cantidad de feromona y dist(i,j) es la distancia del arco (i,j).

Es decir, **la probabilidad es más grande cuando hay más feromonas y menor es la distancia**.

2. Depósito y evaporación

La cantidad de feromonas se actualiza a fin de que los mejores componentes de solución sean más deseables para las hormigas en las futuras iteraciones.

- **Depósito**. Agrega feromonas a los componentes de solución elegidos por las hormigas. Los mejores componentes reciben más feromonas.
- **Evaporación**. Decrece a lo largo del tiempo las feromonas depositadas por hormigas anteriores.

📝 Ejemplo – TSP

La cantidad de feromonas $\tau(i,j)$ de cada arco (i,j) se actualiza:

$$\tau(i,j) \leftarrow (1-\rho).\tau(i,j) + \sum_{1 \leq k \leq m:\ (i,j)\ \in\ tour_k} \frac{1}{dist(tour_k)}$$

donde $\rho \in (0,1)$ y $dist(tour_k)$ es el largo del tour encontrado por la k-ésima hormiga.

Es decir, se evapora el ρ% de la feromona acumulada y, por cada aparición de la arista en el tour de una hormiga, se deposita una cantidad de feromona inversamente proporcional al costo del tour.

Así, **aristas usadas por muchas hormigas y contenidas en tours cortos reciben más feromonas y por ende son más probables a ser elegidas en futuras iteraciones**.

Optimización por Enjambre de Partículas

Particle Swarm Optimization (PSO)

La **optimización por enjambre de partículas** se inspira en el comportamiento de una bandada de aves, un cardumen de peces o un enjambre de insectos.

Un **enjambre** es una colección (población) aparentemente desorganizada de individuos en movimiento que tienden a agruparse, mientras cada individuo parece moverse en una dirección aleatoria.

🐦 Optimización por Enjambre de Partículas

- Se utiliza para minimizar/maximizar una **función de fitness**

$$f : \mathbb{R}^n \to \mathbb{R}$$

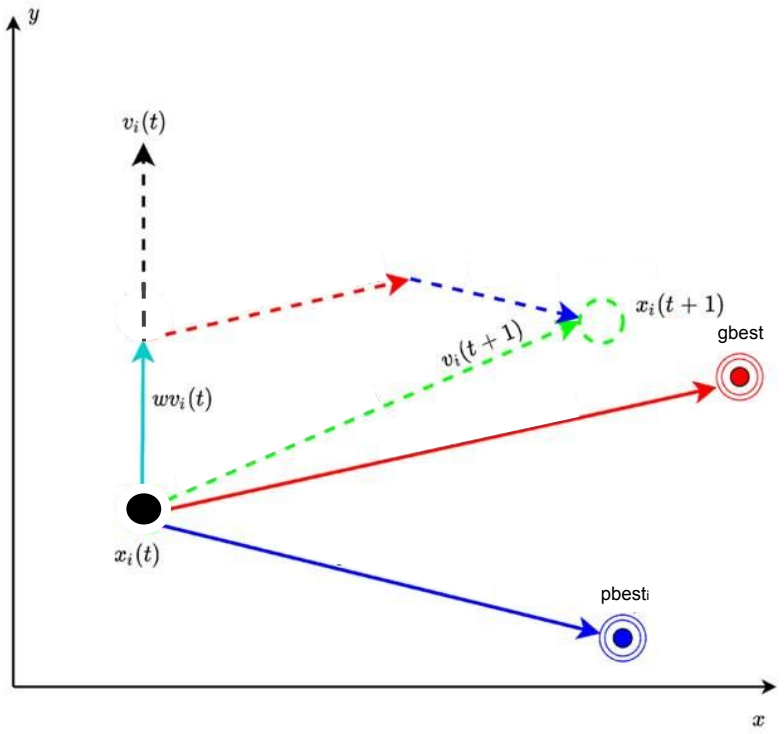
  cuyo gradiente es desconocido.

- La población inicial se compone de un conjunto de **partículas** (puntos de $\mathbb{R}^n$) que inicialmente tienen **posición** y **velocidad** aleatorias.
- Cada partícula recuerda su mejor **posición personal** (pbest) y la mejor **posición global** (gbest) según la función de fitness.
- En cada iteración, cada partícula se acelera estocásticamente hacia pbest y gbest.

Algoritmo de PSO

```
1 function PSO(n, fitness) return partícula
2    iniciar posiciones y velocidades aleatorias para las n partículas
3    iniciar pbesti como la posición inicial de cada partícula i
4    inicial gbest como el pbesti que minimice fitness
5    while no se cumple el criterio de parada do
6         actualizar la velocidad de cada partícula
7         actualizar la posición de cada partícula
8         actualizar pbesti y gbest, si corresponde
9    return gbest
```

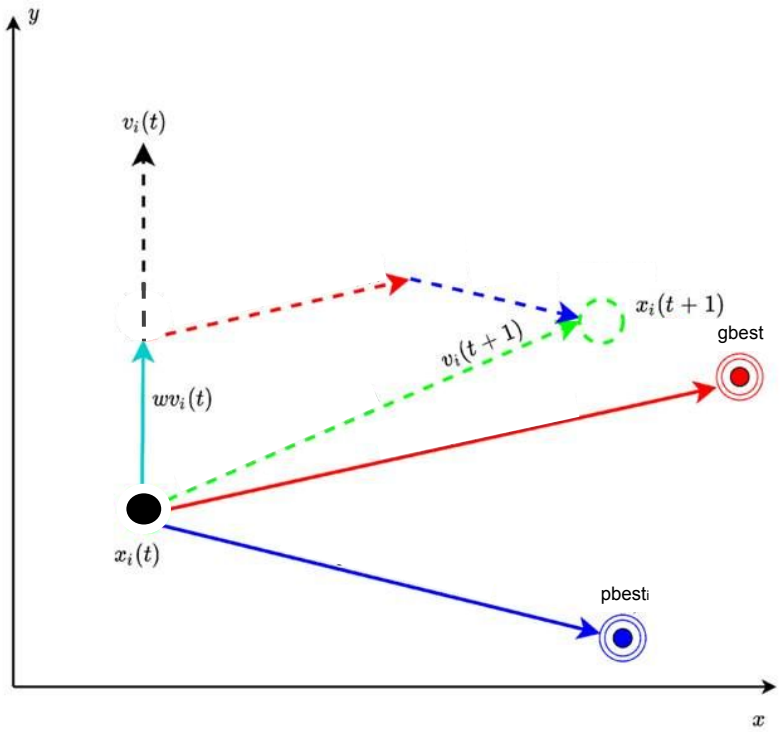
Aceleración de partículas



$$v_i(t+1) = w.v_i(t) + c_1.rand().(gbest - x_i(t)) + c_2.rand().(pbest_i - x_i(t))$$

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

Aceleración de partículas



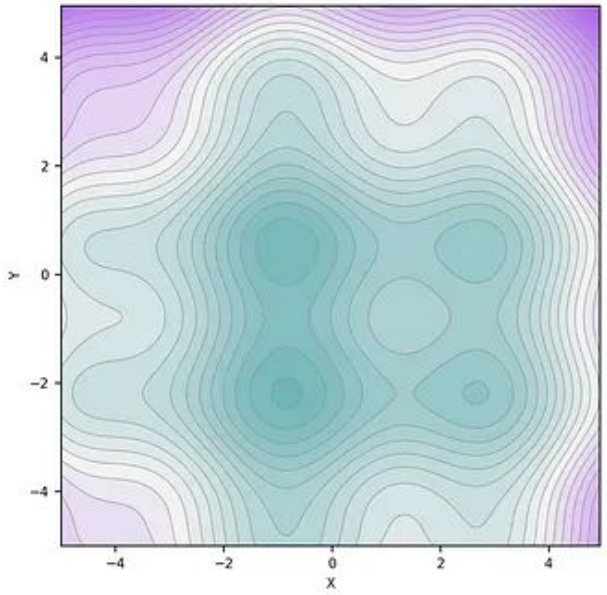
En cada iteración, la **velocidad** de la partícula i se acelera sumando:

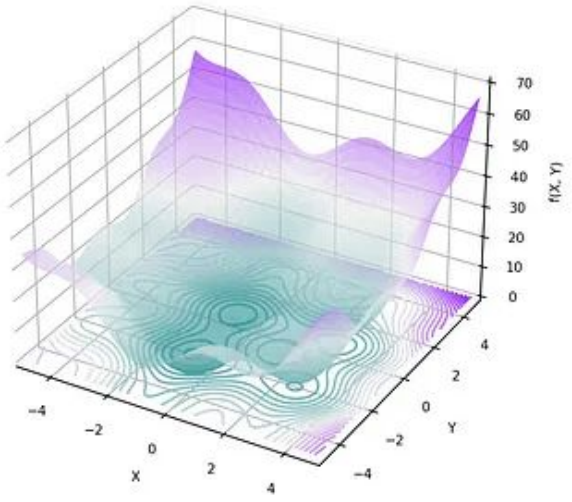
- **Vector celeste**. Su velocidad multiplicada por un factor de inercia.
- **Vector rojo en línea de puntos**. Vector que apunta a gbest multiplicado por un número aleatorio y una constante c_1.
- **Vector azul en línea de puntos**. Vector que apunta a pbest multiplicado por un número aleatorio y una constante c_2.

La **posición** de la partícula i se actualiza sumando su última posición con el **vector verde en línea de puntos** (nueva velocidad).

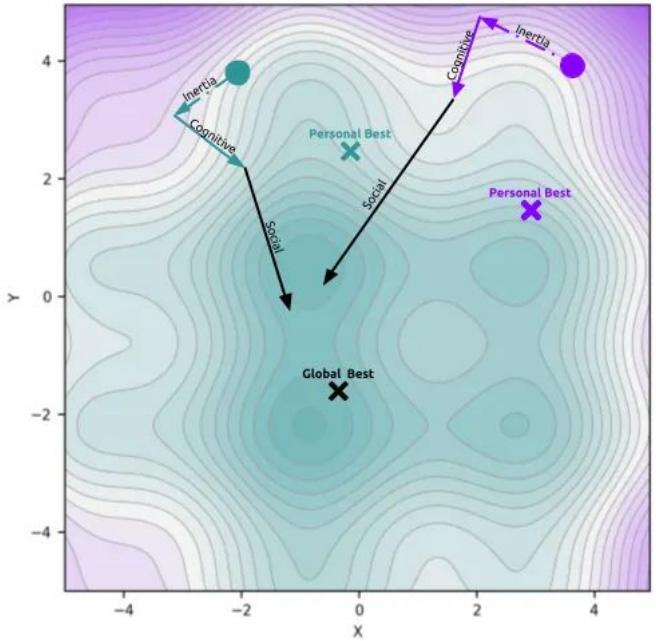
Ejemplos

$$f(x, y) = x^2 + (y + 1)^2 - 5cos(\frac{3}{2}x + \frac{3}{2}) - 3cos(2y - \frac{3}{2})$$





📝 Ejemplos



Link 1

Link 2

Resumen

- Existe una amplia gama de algoritmos de búsqueda inspirados en la **naturaleza**.
- Gran impacto en la resolución de grandes problemas de optimización.
- Muchos de ellos incorporan mecánicas **estocásticas**. Por ejemplo, el **recocido simulado** es una búsqueda local que permite moverse aleatoriamente a soluciones peores, pero esto es cada vez menos **probable** a medida que avanzan las iteraciones.
- Los **algoritmos genéticos**, **la optimización por colonia de hormigas** y **la optimización por enjambre de partículas** son búsquedas locales basadas en **poblaciones**.
- La optimización por enjambre de partículas se utiliza para optimizar funciones reales **no lineales**.