



# Optimización por Enjambres de Partículas PSO

# Introducción

La **optimización por enjambre de partículas** (PSO, «particle swarm optimization») hace referencia a una serie de métodos y algoritmos de optimización heurísticos que evocan el comportamiento de los enjambres de abejas en la naturaleza.



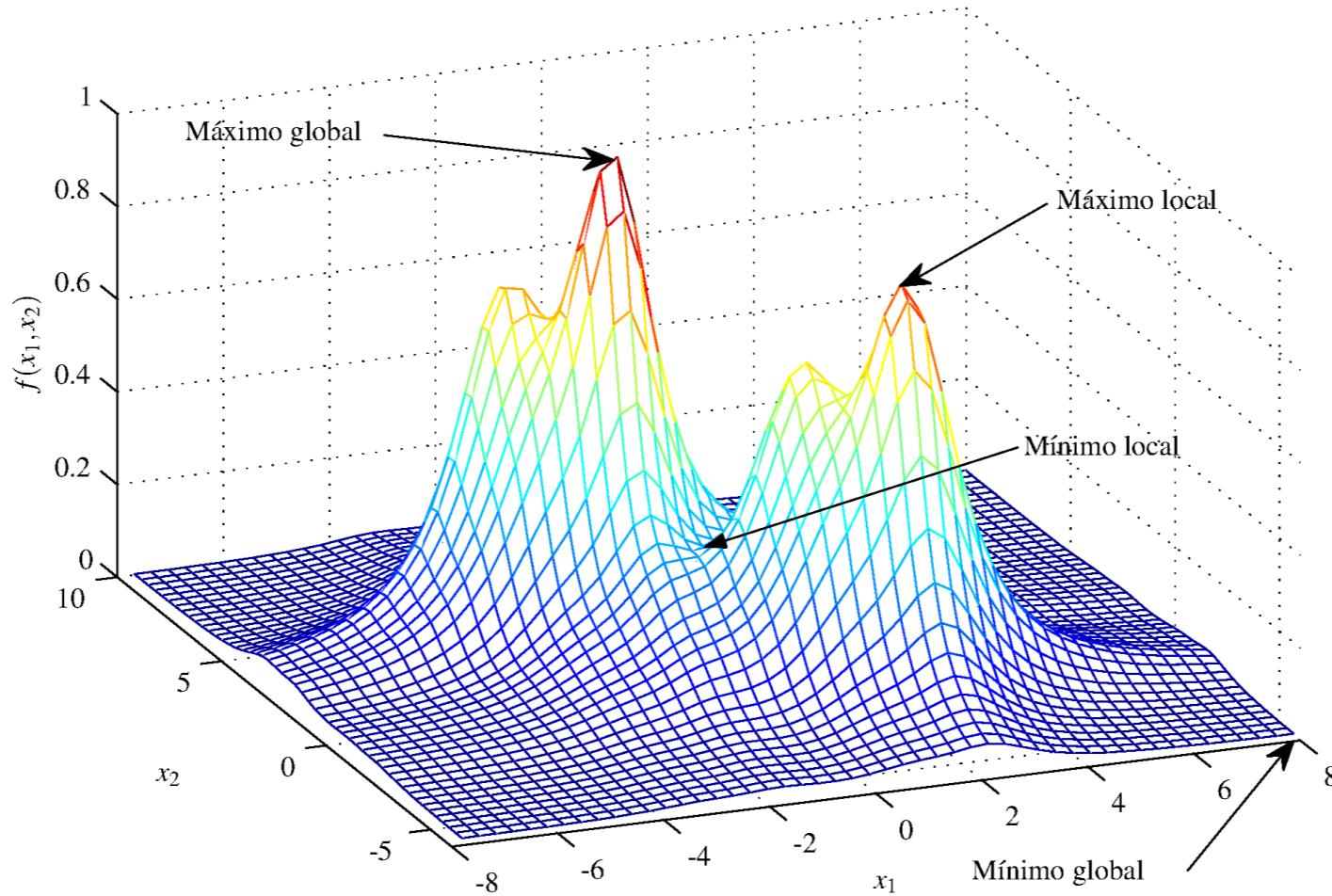
# Introducción

Los métodos **PSO** se atribuyen originalmente a los investigadores *Kennedy, Eberhart* y *Shi*. En un principio fueron concebidos para elaborar modelos de conductas sociales, como el movimiento descrito por los organismos vivos en una bandada de aves o un banco de peces.



# Introducción

Posteriormente el algoritmo se simplificó y se comprobó que era adecuado para problemas de optimización.



# Introducción

Cooperación (PSO) vs. Competitividad (GA)



# PSO

PSO es una **metaheurística**, ya que asume pocas o ninguna hipótesis sobre el problema a optimizar y puede aplicarse en grandes espacios de **soluciones candidatas**. Sin embargo, como toda metaheurística, PSO no garantiza la obtención de la solución óptima en todos los casos.

# PSO

Un algoritmo **PSO** trabaja con una **población** (llamada **nube** o **enjambre**) de **soluciones candidatas** (llamadas **partículas**). Dichas partículas se desplazan a lo largo del **espacio de búsqueda** conforme a ciertas reglas matemáticas. El movimiento de cada partícula depende de su **mejor posición** obtenida, así como de la **mejor posición global** hallada en todo el espacio de búsqueda. A medida que se descubren nuevas y mejores posiciones, éstas pasan a orientar los movimientos de las partículas. El proceso se repite con el objetivo, no garantizado, de hallar en algún momento una solución lo **suficientemente satisfactoria**.

# PSO

Lo descrito anteriormente puede formalizarse del siguiente modo:  
sea  $f : R^n \rightarrow R$  la **función de coste** que se desea minimizar. La función  $f$  toma como argumento una solución candidata, representada como un vector de números reales, y da como salida un número real que indica el valor de la **función objetivo** para la solución candidata obtenida.

# PSO

Las mejores posiciones se corresponden con los mejores valores de la función objetivo  $f$ . El objetivo es hallar una solución  $a$  que verifique  $f(a) \leq f(b)$  para todo  $b$  en el espacio de búsqueda, lo que implicaría que  $a$  es el mínimo global.

# PSO

Sea  $S$  el número de partículas en la nube, cada una de las cuales tiene una posición  $\mathbf{x}_i \in \mathbb{R}^n$  en el espacio de búsqueda y una velocidad  $\mathbf{v}_i \in \mathbb{R}^n$ . Sea  $\mathbf{p}_i$  la mejor posición conocida de una partícula  $i$ , y  $\mathbf{g}$  la mejor posición global conocida. Un algoritmo PSO básico podría describirse como sigue:

# PSO

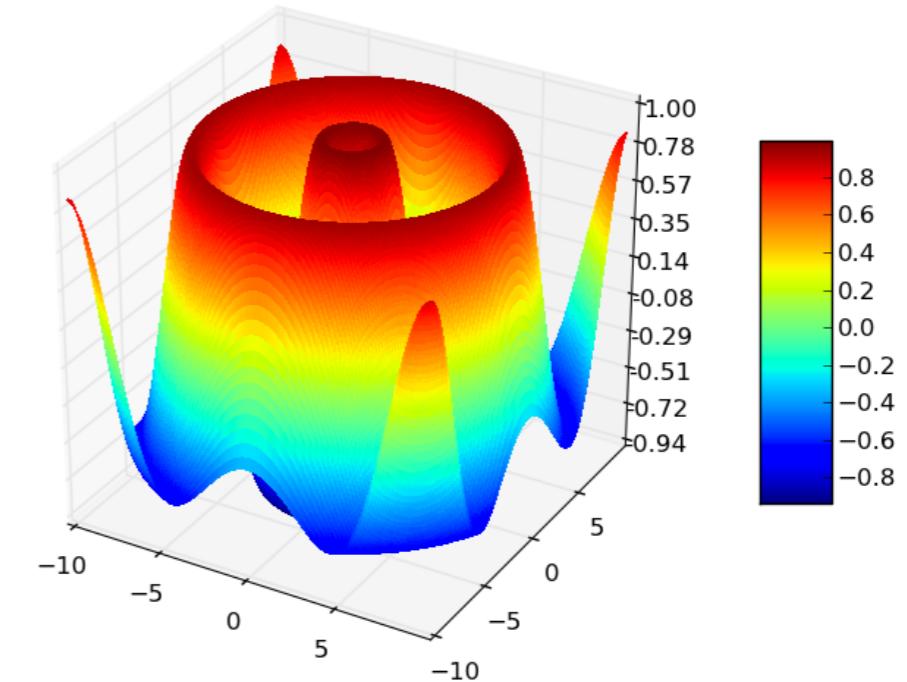
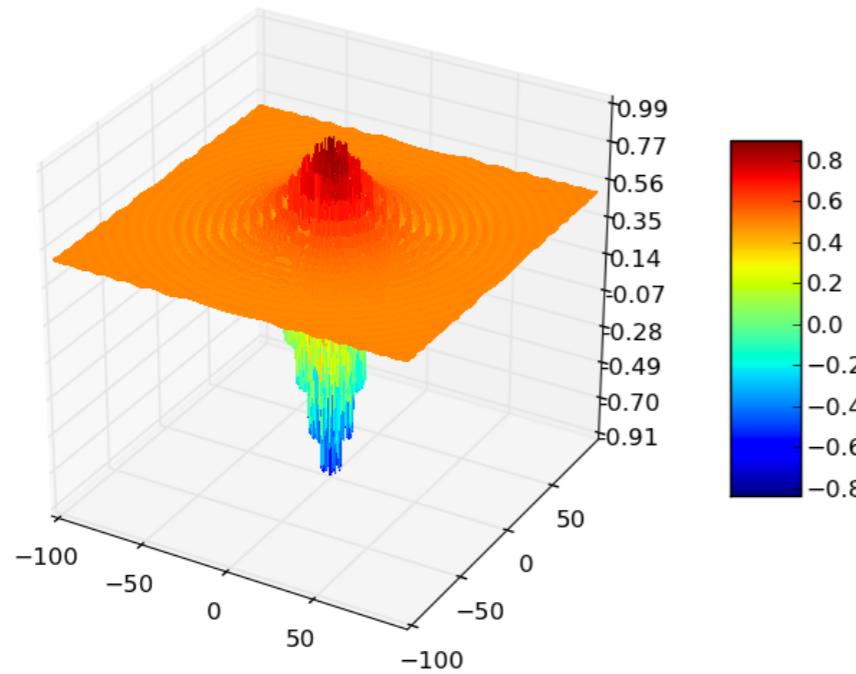
```
for i from 1 to particle_count:  
    particle = new Particle()  
    particles.add(particle)  
    # Randomize particle initial state  
    for j from 0 to param_count-1:  
        # Set particle velocities to random  
        particle.v[j] = random_uniform(0,1)  
        # Set particle parameters to random  
        particle.param[j] = random_uniform(0,1)  
        # Set particle best to match the weights  
        particle.pbest[j] = particle.param[j]  
  
    best_score = min_float  
    # Main loop  
    while best_score<required_score:  
        for each particle in particles:  
            score = score_function(particle)  
            # Update the best particle best  
            if score > particle.best_score:  
                particle.best_score = score  
                particle.pbest = particle.param.clone()  
            # Update global best  
            if score>best_score:  
                best_score = score  
                gbest = particle.param.clone()  
        # Move the particles  
        for each p in particles:  
            for j from 0 to param_count-1:  
                p.v[j] = p.v[j] +  
                    c1 * random uniform() * (p.pbest[j] - p.param[j])  
                    + c2 * random_uniform() * (gbest[j] - p.parms[j])  
                p.param[j] = p.param[j] + p.v[j]
```

# PSO

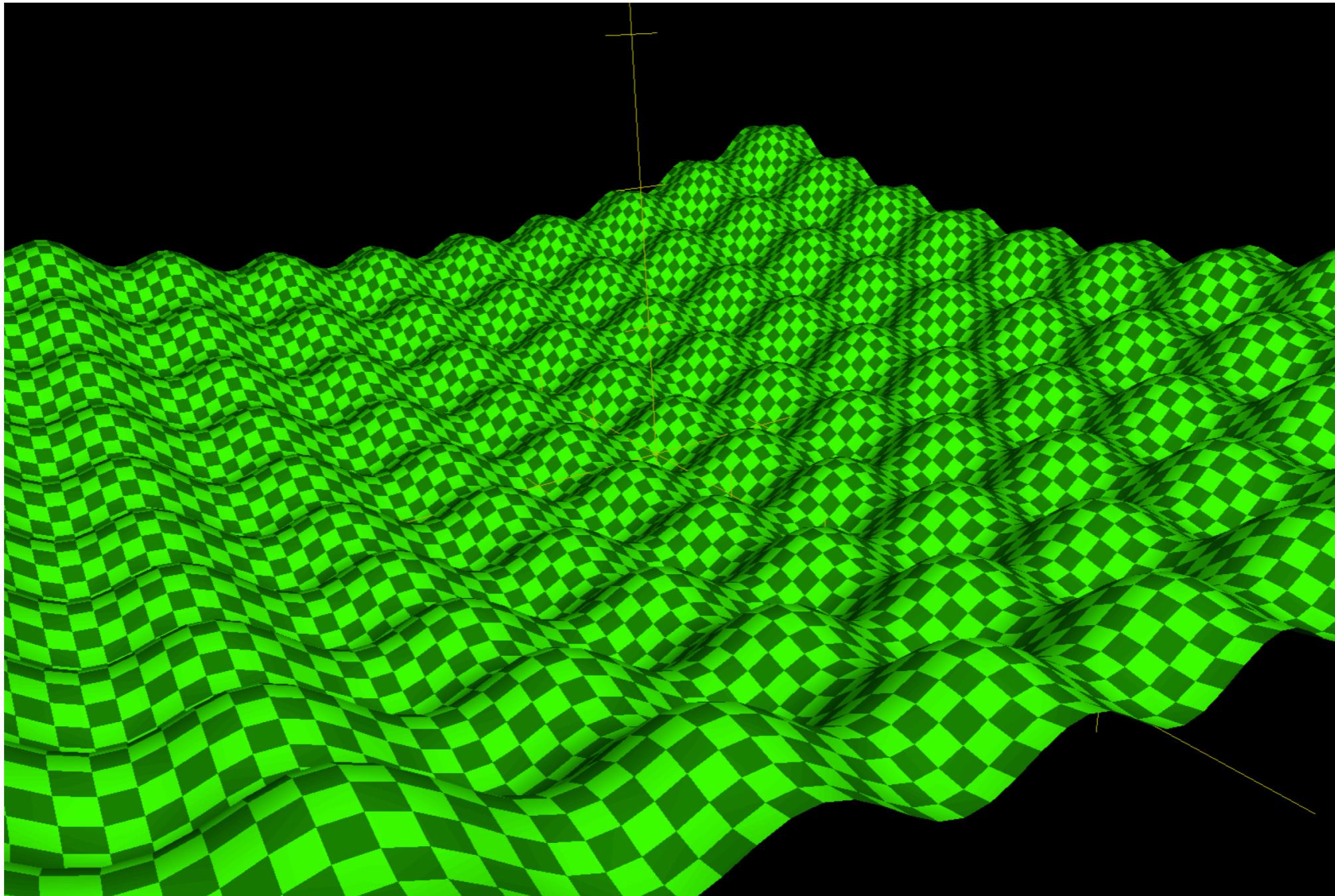
Ejemplo en Python de la función F6 de Schaffer.

$$f(x, y) = 0.5 + \frac{\sin^2(\sqrt{x^2 + y^2}) - 0.5}{[1 + 0.001 \cdot (x^2 + y^2)]^2}$$

que tiene su mínimo en  $f(0, \dots, 0) = 0$



# PSO



# PSO



# PSO

This is a **Particle Swarm Optimisation (PSO)** implementation.

The **objective function** is simply the sum of the squared **x** and squared **y** coordinates of the particle.

$$x^2 + y^2 = cost$$

The **goal** here is to **minimise the objective function score**, therefore the optimal solution is at (0, 0).

**Parameters used:**

c\_1 = 2.0  
c\_2 = 2.0  
max\_velocity = [0.5, 0.5]  
max\_population = 20

```
1. login
[~/Dropbox/proj/toys/playground] > vim bash
```

A close-up photograph of several ants on a red raspberry in a green environment. One ant is carrying a large piece of the raspberry on its back, while others are nearby on the fruit or the surrounding green leaves and moss. The background is blurred green foliage.

# Optimización por Colonia de Hormigas ACO

# Colonia de Hormigas

El primer algoritmo surgió con el objetivo de buscar el **camino óptimo** en un **grafo**, basado en el comportamiento de las hormigas cuando estas están buscando un camino entre la colonia y una fuente de alimentos. La idea original se ha diversificado para resolver una amplia clase de problemas numéricos, y como resultado, han surgido gran cantidad de problemas nuevos, basándose en diversos aspectos del comportamiento de las hormigas.



# Colonia de Hormigas

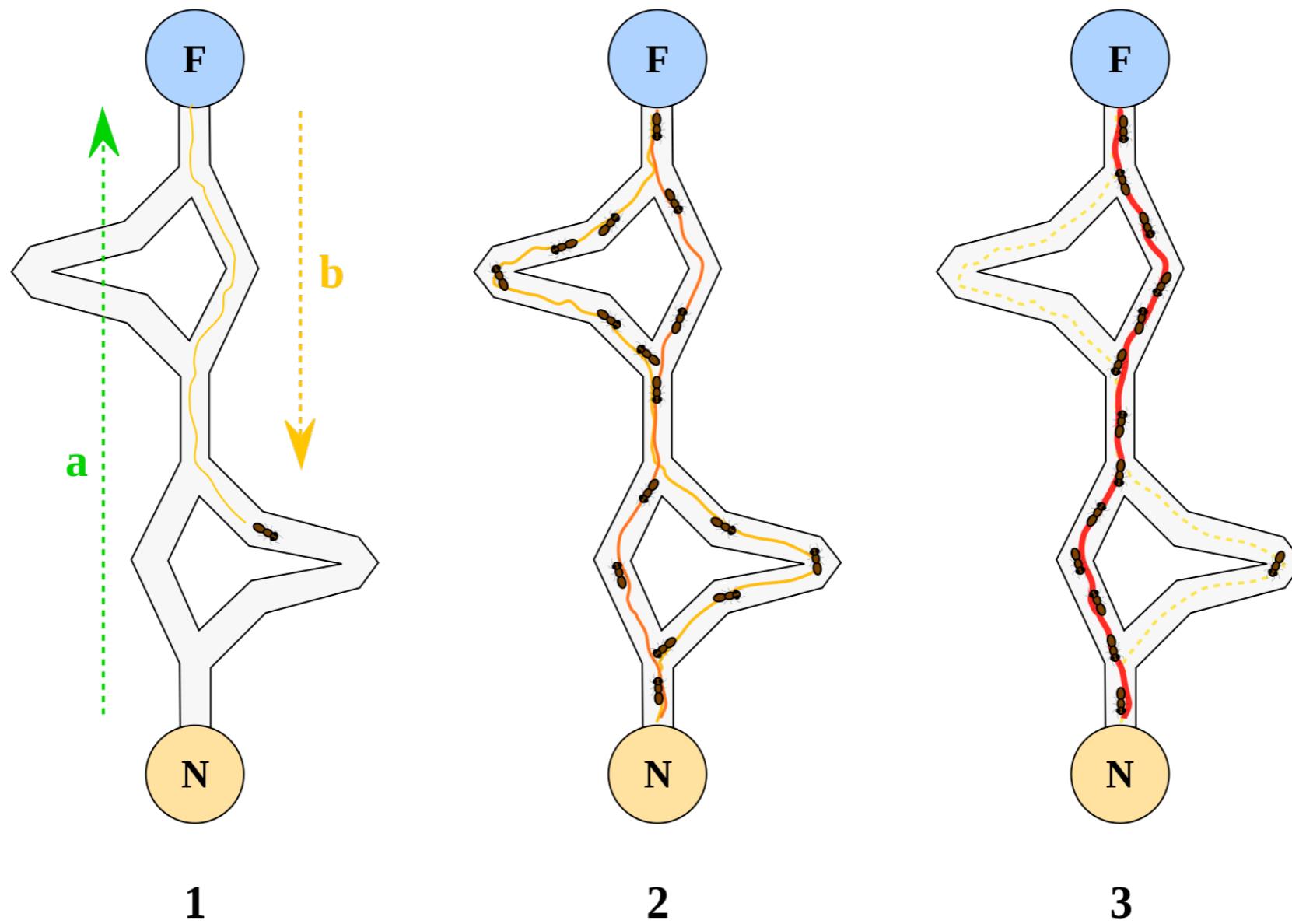
En nuestro mundo natural, las hormigas (initialmente) vagan de manera aleatoria, al azar, y una vez encontrada comida regresan a su colonia dejando un rastro de **feromonas**. Si otras hormigas encuentran dicho **rastro**, es probable que estas no sigan caminando aleatoriamente, puede que estas sigan el rastro de feromonas, regresando y reforzándolo si estas encuentran comida finalmente.



# Colonia de Hormigas

Sin embargo, al paso del tiempo el rastro de feromonas comienza a **evaporarse**, reduciéndose así su fuerza de atracción. Cuanto más tiempo le tome a una hormiga viajar por el camino y regresar de vuelta otra vez, más tiempo tienen las feromonas para evaporarse.

# Colonia de Hormigas



# Colonia de Hormigas

- **ant\_count:** Número de hormigas. Por defecto 30.
- **alpha:** Atracción del camino de feromonas. Por defecto, 1
- **beta:** Atracción de la transición con mejor estado. Por defecto, 5
- **q:** Esta constante establece la cantidad de feromonas que los nodos de un camino comparten en un viaje.
- **initial\_pheromone:** este término es el valor inicial del camino de feromonas. Por defecto, 1.
- **pr:** esta constante define la probabilidad de que una hormiga simplemente llegue algún nodo. Por defecto es 0.01

# Colonia de Hormigas

## Selección de nodo

Para una hormiga  $k$ , la probabilidad  $p_{xy}^k$  de moverse de un estado  $x$  a un estado  $y$  depende de la combinación de dos valores, de el atractivo  $\eta_{xy}$  del movimiento, computado por alguna **heurística** que indica a priori la conveniencia de dicho movimiento y el nivel de **rastro**  $\tau_{xy}$  del movimiento, indicando que tan competente ha sido en el pasado este en particular movimiento.

$$p_{xy}^k = \frac{(\tau_{xy}^\alpha)(\eta_{xy}^\beta)}{\sum_{y \in \text{allowed}_y} (\tau_{xy}^\alpha)(\eta_{xy}^\beta)}$$

# Colonia de Hormigas

## Actualización de feromonas

$$\Delta\tau_{xy}^k = \begin{cases} Q/L_k & \text{si la hormiga } k \text{ usa la trayectoria } xy \text{ en su camino} \\ 0 & \text{en otro caso} \end{cases}$$

donde  $L_k$  es el costo de la ruta de la hormiga  $k$  (en la mayoría de los casos es la longitud) y  $Q$  es una constante.

# Colonia de Hormigas

## Depósito de feromonas

Cuando todas las hormigas han completado un solución, los rastros son actualizados por:

$$\tau_{xy} = \rho\tau_{xy} + \sum_k \Delta\tau_{xy}^k$$

donde  $\tau_{xy}$  es la cantidad de feromonas depositadas para un estado de transición  $xy$ ,  $\rho$  es el coeficiente de evaporación de feromonas y  $\Delta\tau_{xy}^k$  es la cantidad de feromonas depositadas por la hormiga  $k$ .

A photograph showing a person's gloved hands holding a long, thin, glowing orange-red glass tube. The tube is being heated over a bright orange-red furnace. The background is dark, making the glowing glass stand out.

Simulated Annealing  
SA

# Simulated Annealing

Es un algoritmo de búsqueda meta-heurística para problemas de optimización global; el objetivo general de este tipo de algoritmos es encontrar una buena aproximación al valor óptimo de una función en un espacio de búsqueda grande.

# Simulated Annealing

¿Qué significa “simulated annealing”? “Annealing” es el proceso de calentar y enfriar materiales para modificar sus propiedades físicas mediante la alteración de su estructura interna.

Durante el “simulated annealing” mantenemos una temperatura variable para simular este proceso de calentamiento. Inicialmente la temperatura es alta y, a medida que avanza el algoritmo, esta va bajando.

Mientras la temperatura es alta, el algoritmo va permitiendo con mayor frecuencia soluciones peores que la actual. Dando la oportunidad de saltar lejos de un mínimo local.

# Simulated Annealing

En el caso de un algoritmo “hill climber” éste solo permitirá soluciones mejores que la actual. “Simulated Annealing” evita este problema.



# Simulated Annealing

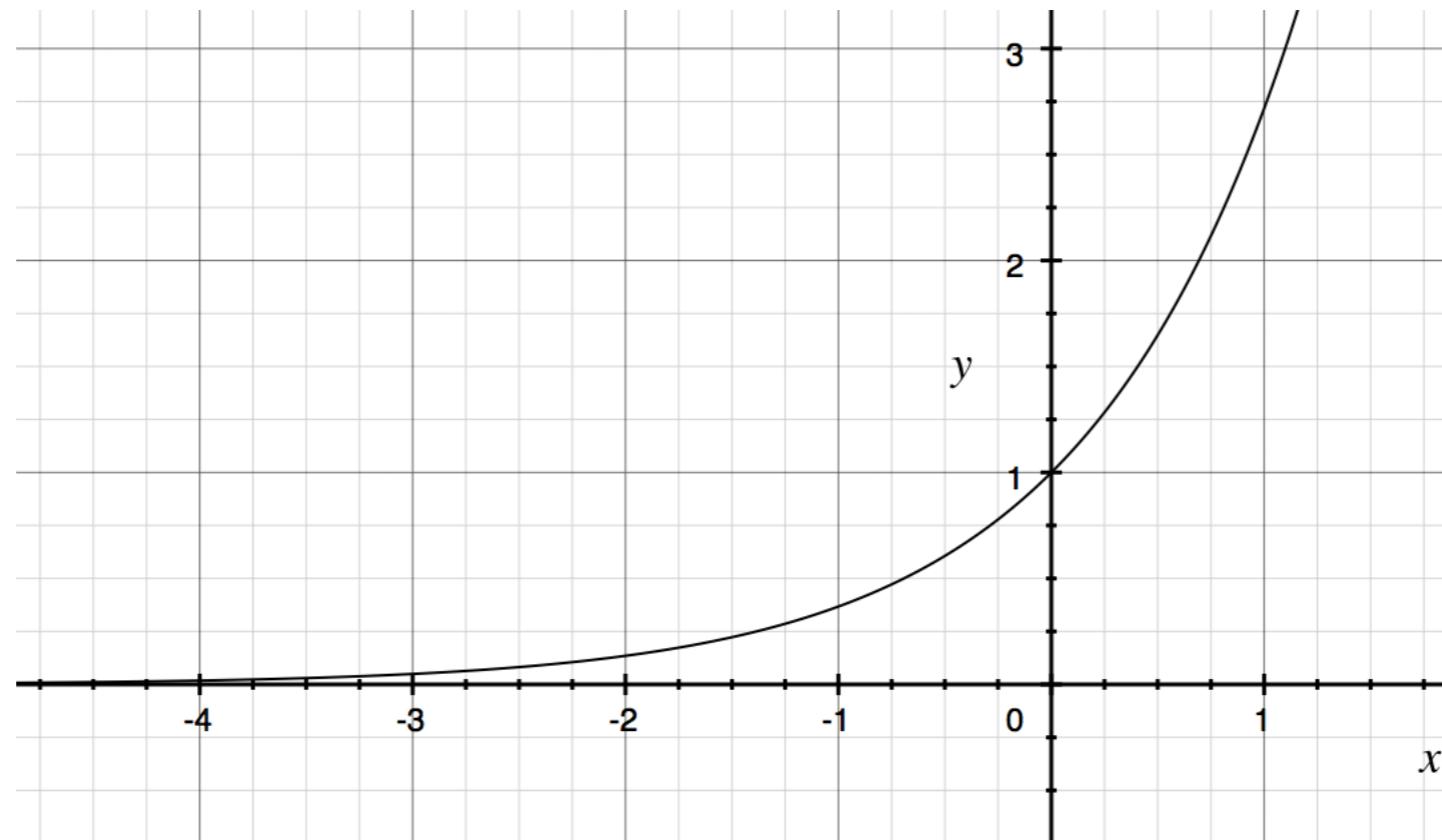
## Función de aceptación

Comprobamos si la solución vecina es mejor que la actual. Si es así la aceptamos incondicionalmente. Si no, consideraremos un par de factores. Primero, cómo de peor es y, segundo, cómo de alta es la temperatura actual.

$$e^{\frac{Energia_{solucion} - Energia_{vecino}}{Temperatura}}$$

# Simulated Annealing

## Función de aceptación



$$e^{\frac{Energia_{solucion} - Energia_{vecino}}{Temperatura}}$$

# Simulated Annealing

## Resumen del algoritmo

- Inicialización de la temperatura y se crea una solución aleatoria inicial.
- Iterar hasta llegar a la condición de parada. Se ha llegado al límite inferior de la temperatura o se ha encontrado una solución satisfactoria.
- Se selecciona una solución vecina mediante un pequeño cambio en la solución actual.
- Se decide si moverse a la solución vecina.
- Se decrementa la temperatura y se continúa con la siguiente iteración.