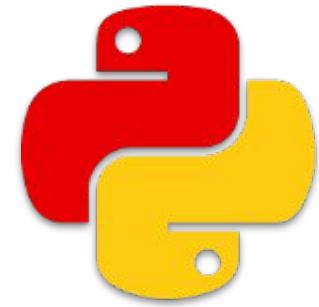


# Introducción a los algoritmos bioinspirados con Python



PyConES 2021

---

Andrea Morales Garzón



@andreamorgar



andreamorgarz@gmail.com

Antonio Manjavacas Lucas

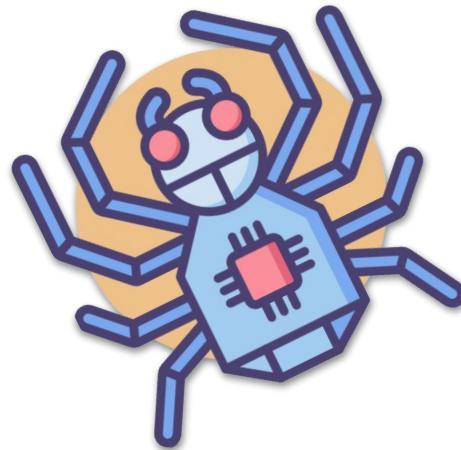


@manjavacas\_



antomanjavacas@gmail.com

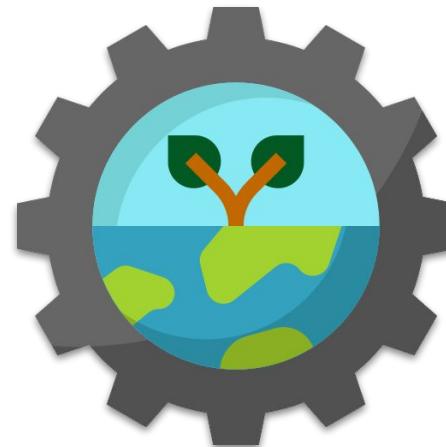
# ALGORITMOS BIOINSPIRADOS



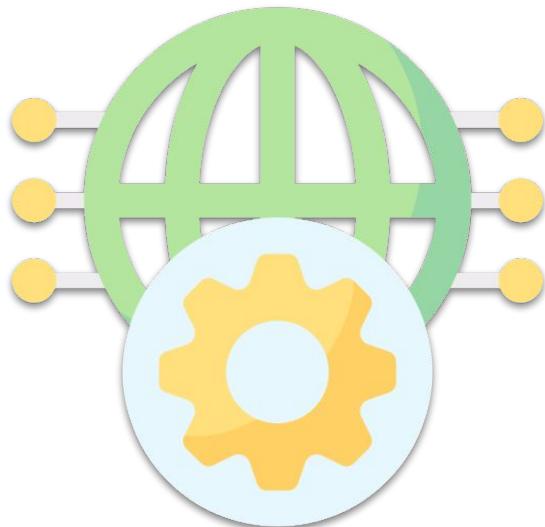
# ¿Qué son los algoritmos bioinspirados?

Algoritmos inspirados en el funcionamiento de sistemas naturales

Modelan de forma aproximada fenómenos existentes en la naturaleza



Permiten resolver problemas de gran **complejidad** computacional



Optimización → búsqueda de la solución óptima a un problema

Problemas orientados a maximizar beneficios, minimizar costes, riesgos, recursos, tiempo, etc.

# Una enorme cantidad de aplicaciones...

- Cálculo de **rutas mínimas** entre dos puntos
- **Planificación** de tareas
- Problemas **multiobjetivo** y/o basados en **restricciones**
- Enrutamiento de **redes de comunicaciones** y **vehículos**
- **Robótica**
- Coloreado de grafos
- **Aprendizaje automático** y **predicción**
- **Medicina**: calibración de equipos, etc.
- Simulación de **sistemas naturales**:
  - ↳ incendios forestales, reacciones químicas, mecánica de fluidos, patrones de pigmentación de la piel, comportamiento de colonias, interacción entre partículas, etc.

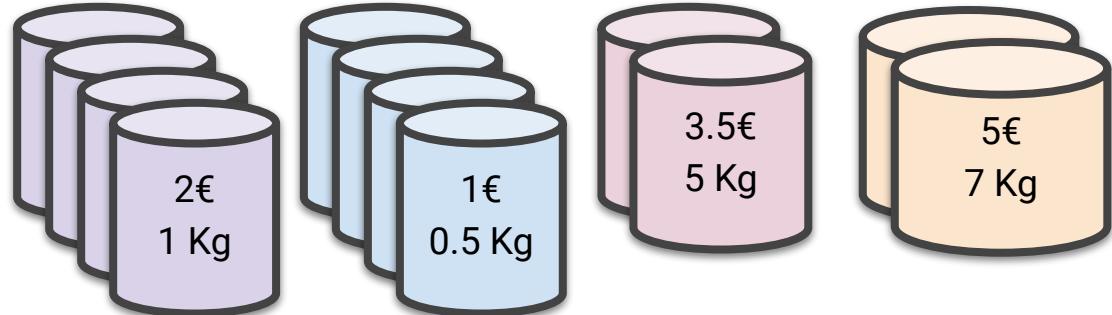


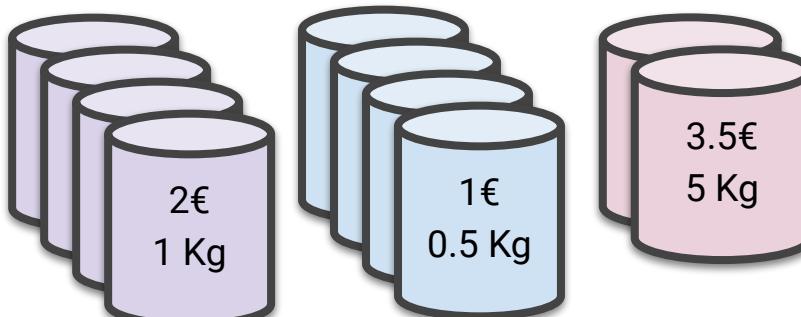
## El problema de la mochila



## The knapsack problem

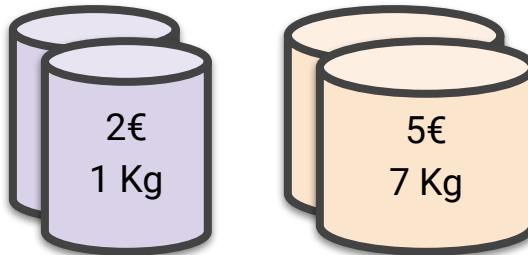
Dada una mochila con capacidad N que puede llenarse con objetos de distinto peso y valor, ¿qué objetos elijo para maximizar las ganancias sin exceder la capacidad permitida?





$$1 \text{ Kg} \times 4 + 0.5 \text{ Kg} \times 4 + 5 \text{ Kg} \times 2 = 16 \text{ Kg} \quad \checkmark$$

$$2 \text{ €} \times 4 + 1 \text{ €} \times 4 + 3.5 \text{ €} \times 2 = 19 \text{ €} \quad 😊$$



$$1 \text{ Kg} \times 2 + 7 \text{ Kg} \times 2 = 16 \text{ Kg} \quad \checkmark$$

$$2 \text{ €} \times 2 + 5 \text{ €} \times 2 = 14 \text{ €} \quad \text{😊}$$

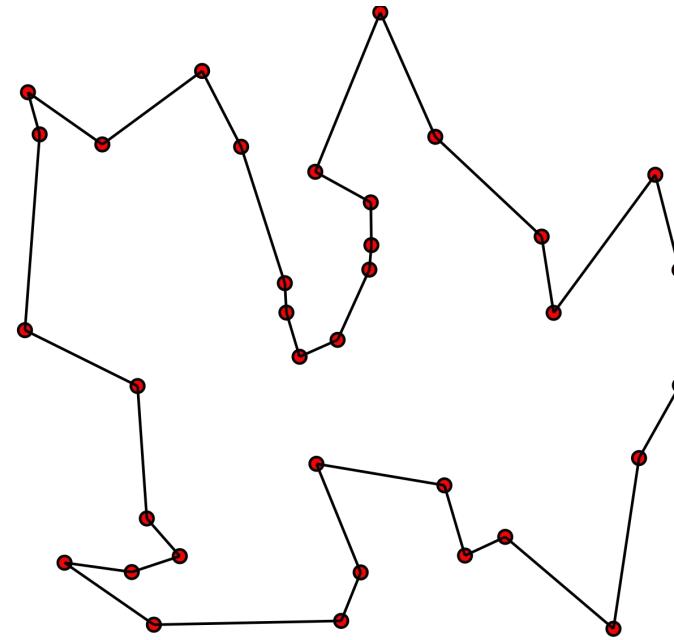
# MY HOBBY: EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS



El **problema** de la mochila es equiparable a múltiples problemas de la misma naturaleza....

Por ejemplo, el **problema del viajante de comercio** (*Travelling Salesman Problem*)

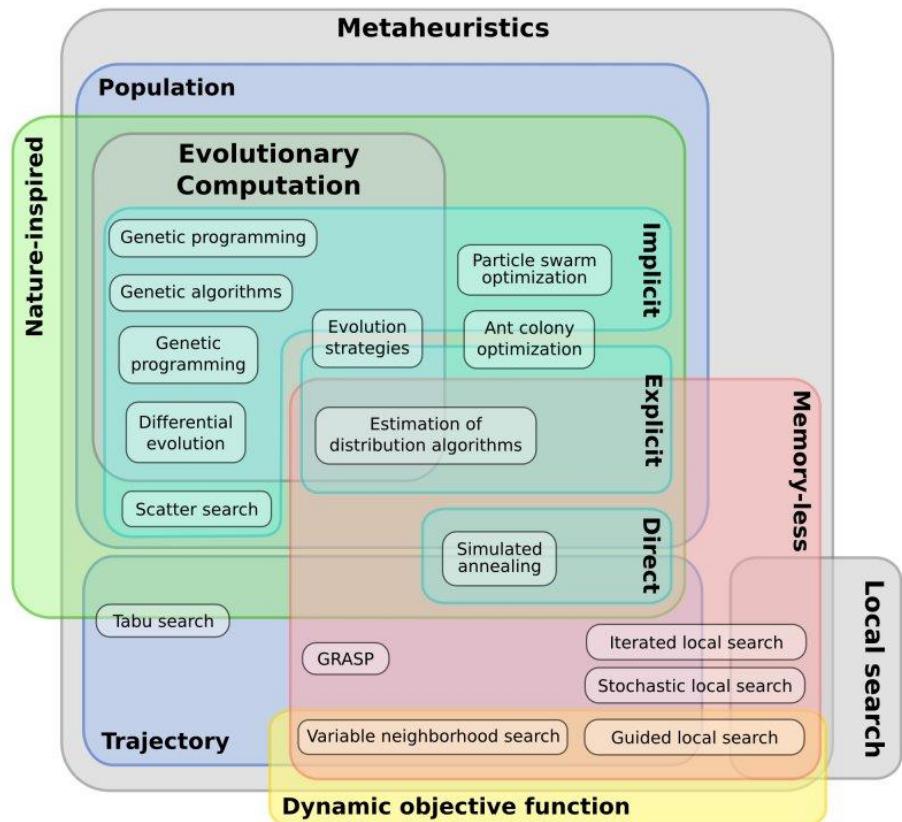
Problemas NP → es más sencillo comprobar si una solución es óptima que obtenerla

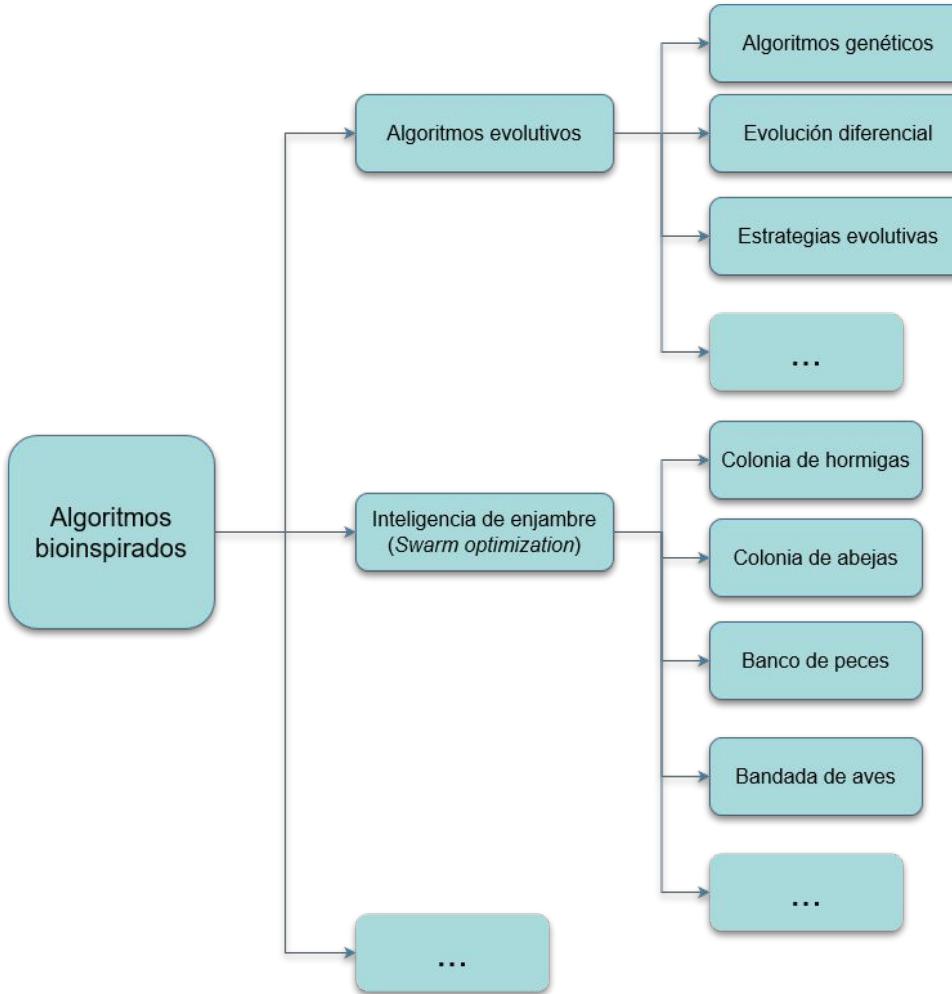


# No obstante...

Los algoritmos bioinspirados  
pueden ayudarnos a resolver  
este tipo de problemas

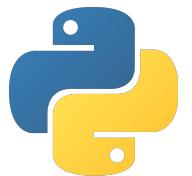
Metaheurísticas → orientadas  
a problemas que **no** cuentan  
con un método de resolución  
óptimo (heurística, algoritmo...)





# inspyred

Bio-inspired Algorithms in Python

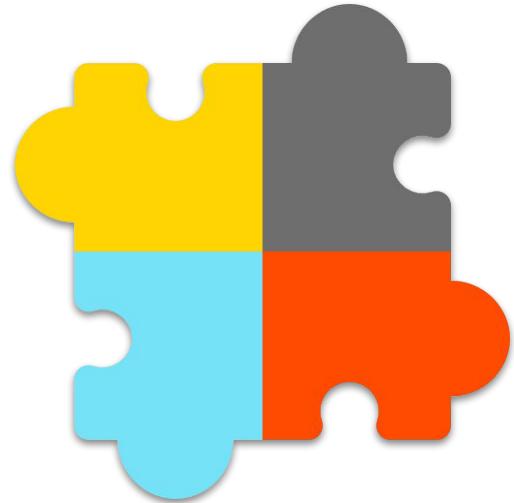


Los algoritmos **bioinspirados** tienen (al menos) dos características dependientes del **problema**:

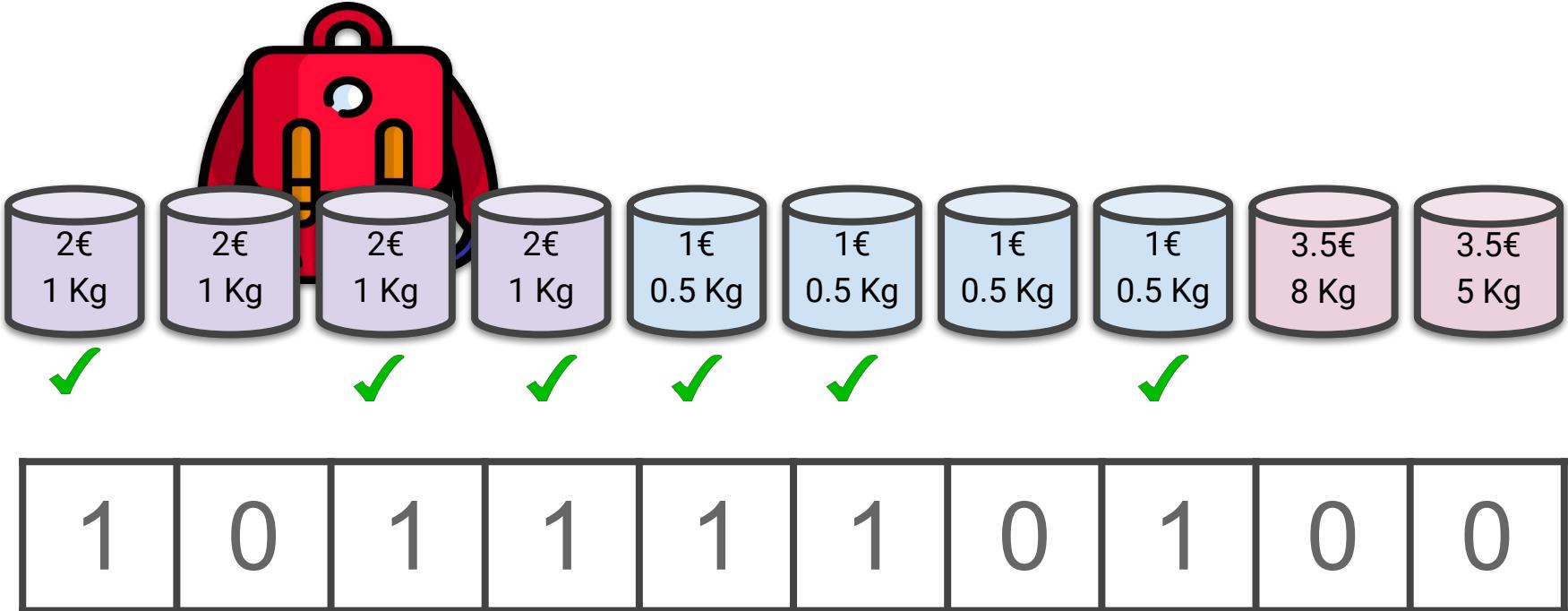
- la **representación** de la solución
- la **evaluación** de la solución

También hay características que dependen del **algoritmo**:

- métodos de selección, reemplazo, funciones para almacenamiento de soluciones, etc.



Objetivo de **inspyred**: separar los **componentes específicos del problema** de aquellos **específicos del algoritmo** para crear algoritmos lo más genéricos posibles



- Vector de tantos elementos como objetos
- Si el valor en la posición  $i$  es 1: el objeto  $i$ -ésimo lo seleccionamos

**inspyred**: computación evolutiva dividida en dos partes...

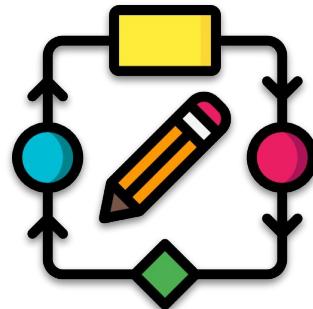
### Componentes específicos del problema

- *Generator*: define cómo se crea una solución
- *Evaluator*: evalúa las soluciones generadas



### Componentes específicos del algoritmo

- *Selector*: decide qué soluciones se convierten en padres de futuras generaciones
- *Migrator*: decide cómo las poblaciones pasan a nuevas generaciones
- ...



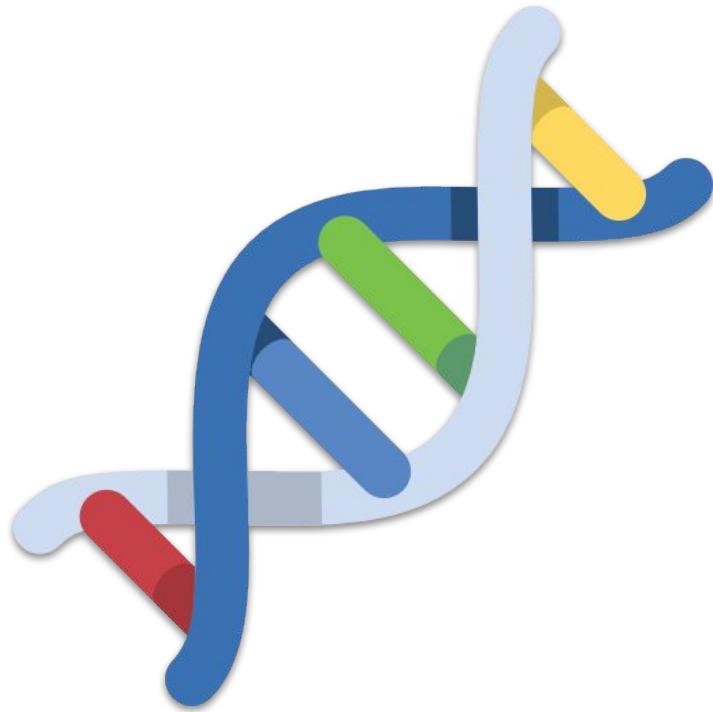
# Algoritmos disponibles en inspyred

# ALGORITMOS GENÉTICOS

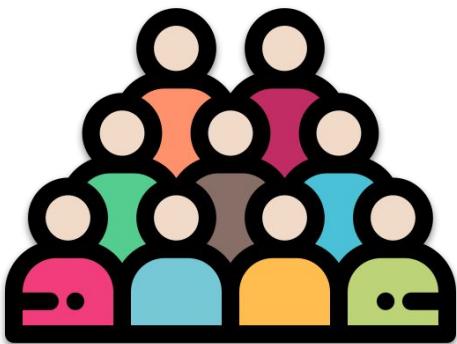
*Genetic algorithms (GA)*

Replican el proceso de evolución natural propuesto por Darwin.

Basados en hacer evolucionar poblaciones de soluciones hacia valores óptimos del problema.



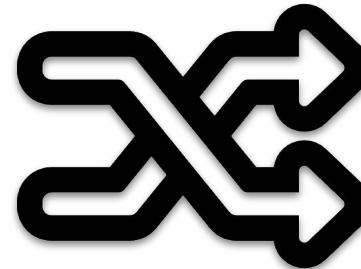
# Operadores



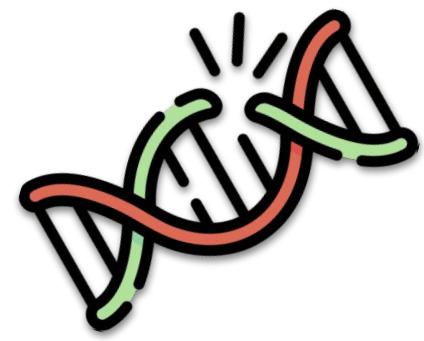
POBLACIÓN  
*soluciones*



EVALUACIÓN  
*fitness*

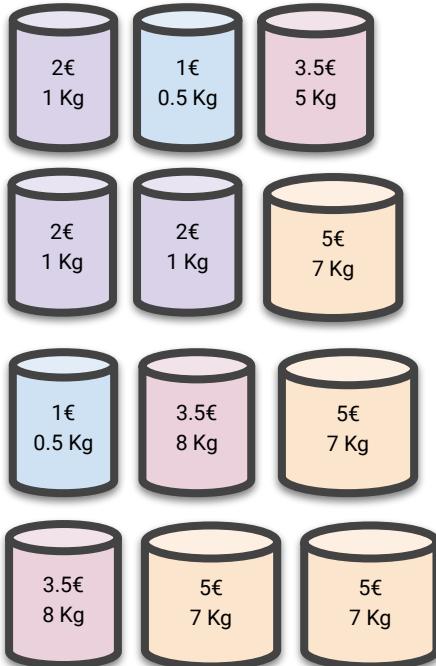


CRUCE

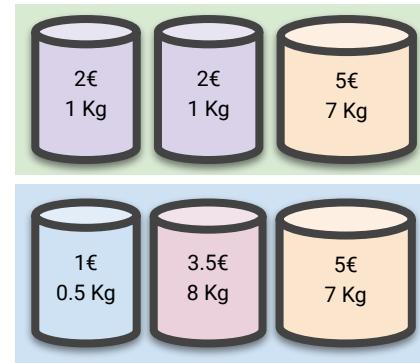


MUTACIÓN  
*aleatoria*

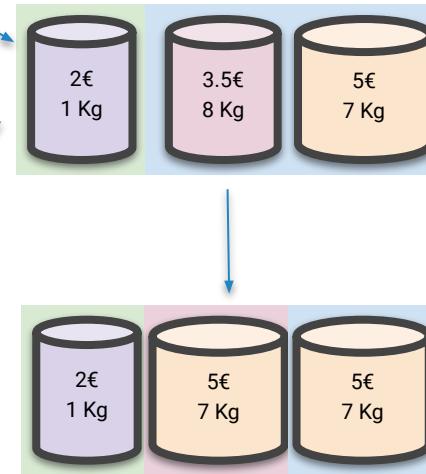
## POBLACIÓN



## SELECCIÓN

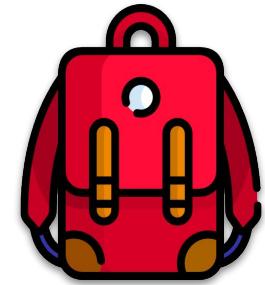


## CRUCE



## REEMPLAZO

## MUTACIÓN



```
ea = inspyred.ec.EvolutionaryComputation(random.Random())
ea.selector = inspyred.ec.selectors.tournament_selection
ea.variator = [inspyred.ec.variators.uniform_crossover,
               inspyred.ec.variators.gaussian_mutation]
ea.replacer = inspyred.ec.replacers.steady_state_replacement
ea.terminator = inspyred.ec.terminators.evaluation_termination

def GA(problem):
    final_pop = ea.evolve(generator=problem.generator,
                           evaluator=problem.evaluator,
                           bounder=problem.bounder,
                           maximize=problem.maximize,
                           pop_size=100,
                           max_evaluations=30000,
                           tournament_size=5,
                           num_selected=2)
    best = max(final_pop)
    print('Best Solution: {0}: {1}'.format(str(best.candidate), best.fitness))

GA(problem_no_duplicates)
GA(problem_duplicates)
```

```

ea = inspyred.ec.EvolutionaryComputation(random.Random())
ea.selector = inspyred.ec.selectors.tournament_selection
ea.variator = [inspyred.ec.variators.uniform_crossover,
               inspyred.ec.variators.gaussian_mutation]
ea.replacer = inspyred.ec.replacers.steady_state_replacement
ea.terminator = inspyred.ec.terminators.evaluation_termination

def GA(problem):
    final_pop = ea.evolve(generator=problem.generator,
                           evaluator=problem.evaluator,
                           bounder=problem.bounder,
                           maximize=problem.maximize,
                           pop_size=100,
                           max_evaluations=30000,
                           tournament_size=5,
                           num_selected=2)
    best = max(final_pop)
    print('Best Solution: {0}: {1}'.format(str(best.candidate), best.fitness))

GA(problem_no_duplicates)
GA(problem_duplicates)

```

Instanciar problema evolutivo

EvolutionaryComputation es una abstracción de cualquier tipo de algoritmo evolutivo

```

ea = inspyred.ec.EvolutionaryComputation(random.Random())

ea.selector = inspyred.ec.selectors.tournament_selection
ea.variator = [inspyred.ec.variators.uniform_crossover,
               inspyred.ec.variators.gaussian_mutation]
ea.replacer = inspyred.ec.replacers.steady_state_replacement
ea.terminator = inspyred.ec.terminators.evaluation_termination

def GA(problem):
    final_pop = ea.evolve(generator=problem.generator,
                          evaluator=problem.evaluator,
                          bounder=problem.bounder,
                          maximize=problem.maximize,
                          pop_size=100,
                          max_evaluations=30000,
                          tournament_size=5,
                          num_selected=2)
    best = max(final_pop)
    print('Best Solution: {0}: {1}'.format(str(best.candidate), best.fitness))

GA(problem_no_duplicates)
GA(problem_duplicates)

```

Definición de los operadores:  
 selección,  
 cruce + mutación,  
 reemplazo y  
 terminación

```
ea = inspyred.ec.EvolutionaryComputation(random.Random())
ea.selector = inspyred.ec.selectors.tournament_selection
ea.variator = [inspyred.ec.variators.uniform_crossover,
               inspyred.ec.variators.gaussian_mutation]
ea.replacer = inspyred.ec.replacers.steady_state_replacement
ea.terminator = inspyred.ec.terminators.evaluation_termination
```

```
def GA(problem):
    final_pop = ea.evolve(generator=problem.generator,
                           evaluator=problem.evaluator,
                           bounder=problem.bounder,
                           maximize=problem.maximize,
                           pop_size=100,
                           max_evaluations=30000,
                           tournament_size=5,
                           num_selected=2)
    best = max(final_pop)
    print('Best Solution: {0}: {1}'.format(str(best.candidate), best.fitness))
```

```
GA(problem_no_duplicates)
GA(problem_duplicates)
```

Definición del  
algoritmo genético

```

ea = inspyred.ec.EvolutionaryComputation(random.Random())
ea.selector = inspyred.ec.selectors.tournament_selection
ea.variator = [inspyred.ec.variators.uniform_crossover,
               inspyred.ec.variators.gaussian_mutation]
ea.replacer = inspyred.ec.replacers.steady_state_replacement
ea.terminator = inspyred.ec.terminators.evaluation_termination

def GA(problem):
    final_pop = ea.evolve(generator=problem.generator,
                           evaluator=problem.evaluator,
                           bounder=problem.bounder,
                           maximize=problem.maximize,
                           pop_size=100,
                           max_evaluations=30000,
                           tournament_size=5,
                           num_selected=2)
    best = max(final_pop)
    print('Best Solution: {0}: {1}'.format(str(best.candidate), best.fitness))

```

GA(problem\_no\_duplicates)  
GA(problem\_duplicates)

Best Solution: [1, 1, 1, 1, 0, 0, 0]: 22  
Best Solution: [3, 0, 4, 0, 0, 0, 0]: 34

Solución

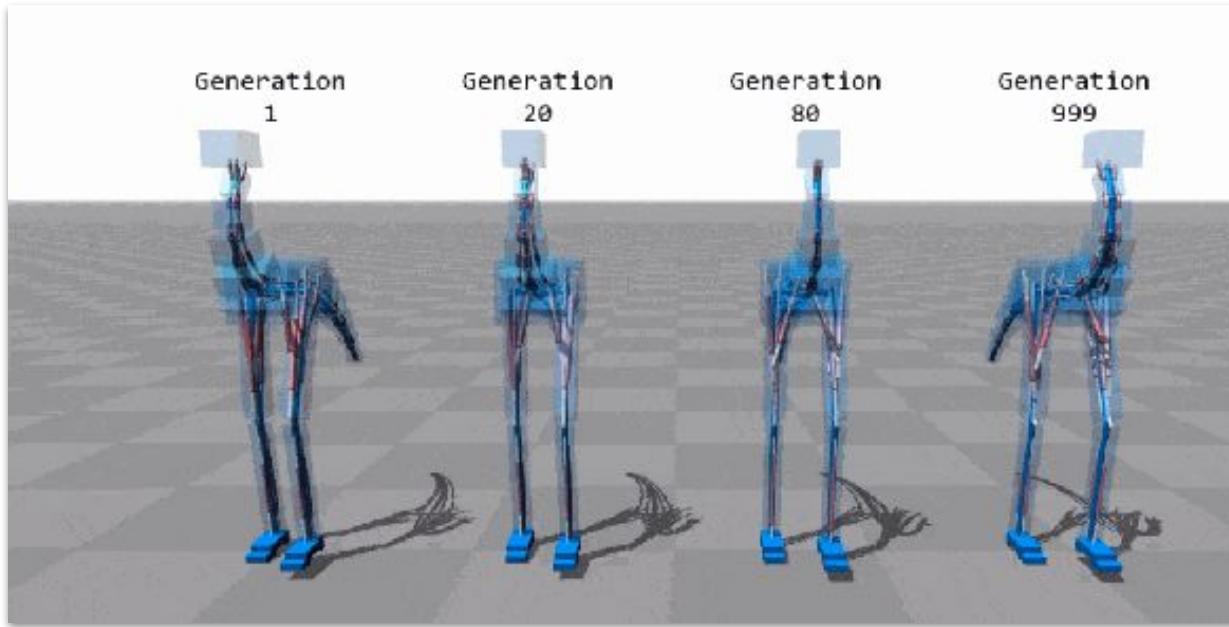
Implementación de  
algoritmo genético  
“canónico”

```
def GA_canonic(problem):  
    ea = inspyred.ec.GA(random.Random())  
    ea.terminator = inspyred.ec.terminators.evaluation_termination  
    final_pop = ea.evolve(generator=problem.generator,  
                          evaluator=problem.evaluator,  
                          pop_size=100,  
                          maximize=problem.maximize,  
                          bounder=problem.bounder,  
                          max_evaluations=30000,  
                          num_elites=1)  
  
    best = max(final_pop)  
    print('Best Solution: \n{}'.format(str(best)))  
  
GA_canonic(problem_no_duplicates)  
GA_canonic(problem_duplicates)
```

Diseño simplificado  
utilizando la clase GA

```
def GA_canonic(problem):  
    ea = inspyred.ec.GA(random.Random())  
    ea.terminator = inspyred.ec.terminators.evaluation_termination  
    final_pop = ea.evolve(generator=problem.generator,  
                          evaluator=problem.evaluator,  
                          pop_size=100,  
                          maximize=problem.maximize,  
                          bounder=problem.bounder,  
                          max_evaluations=30000,  
                          num_elites=1)  
  
    best = max(final_pop)  
    print('Best Solution: \n{}'.format(str(best)))  
  
GA_canonic(problem_no_duplicates)  
GA_canonic(problem_duplicates)
```

Misma filosofía para  
el resto de algoritmos

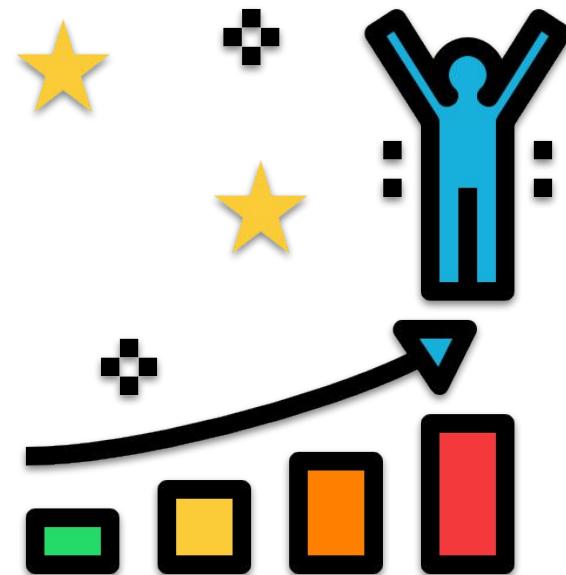


# ESTRATEGIAS EVOLUTIVAS

*Evolution strategies (ES)*

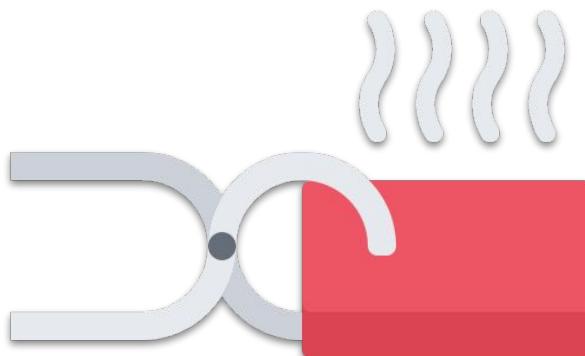
Algoritmo de tipo evolutivo

- Las soluciones son **vectores de números reales**.
- La solución codificada no es **binaria** (0-1), sino que se codifica directamente por sus **valores reales**.
- El operador de **mutación** añade valores a los parámetros siguiendo una **distribución normal multivariante**.



# ENFRIAMIENTO SIMULADO

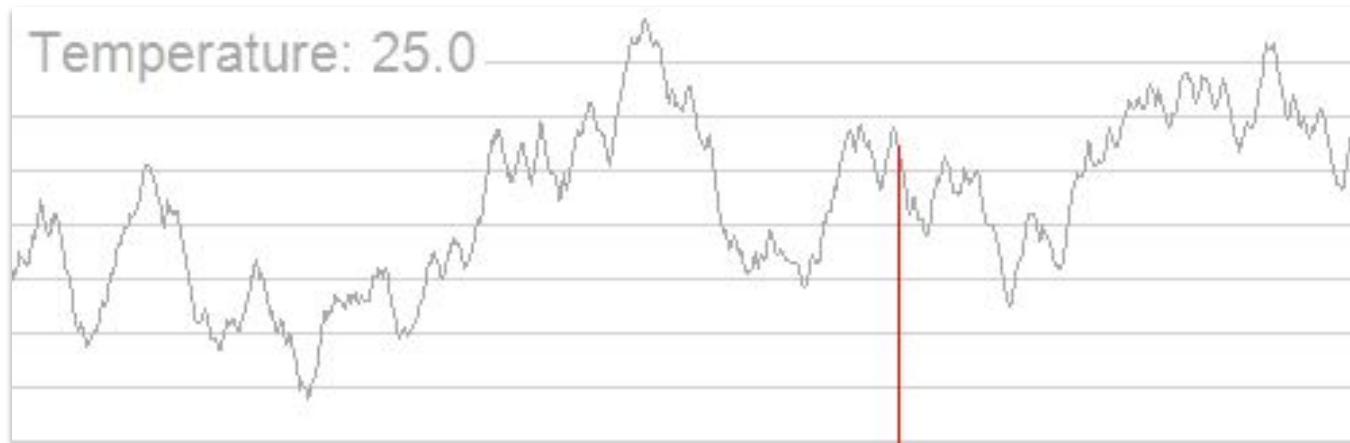
*Temple simulado, recocido simulado, cristalización simulada, simulated annealing (SA)*



- Algoritmo de búsqueda local
- Parte de **soluciones parciales** que se mueven a **soluciones vecinas mejores**.
- Para evitar caer en **óptimos locales**, se permiten transiciones hacia **soluciones peores** de forma controlada.
- Este control viene dado por una **probabilidad decreciente en el tiempo**:

$$p(A) = e^{\left(\frac{valor(A)-valor(A')}{t}\right)}$$

# ENFRIAMIENTO SIMULADO



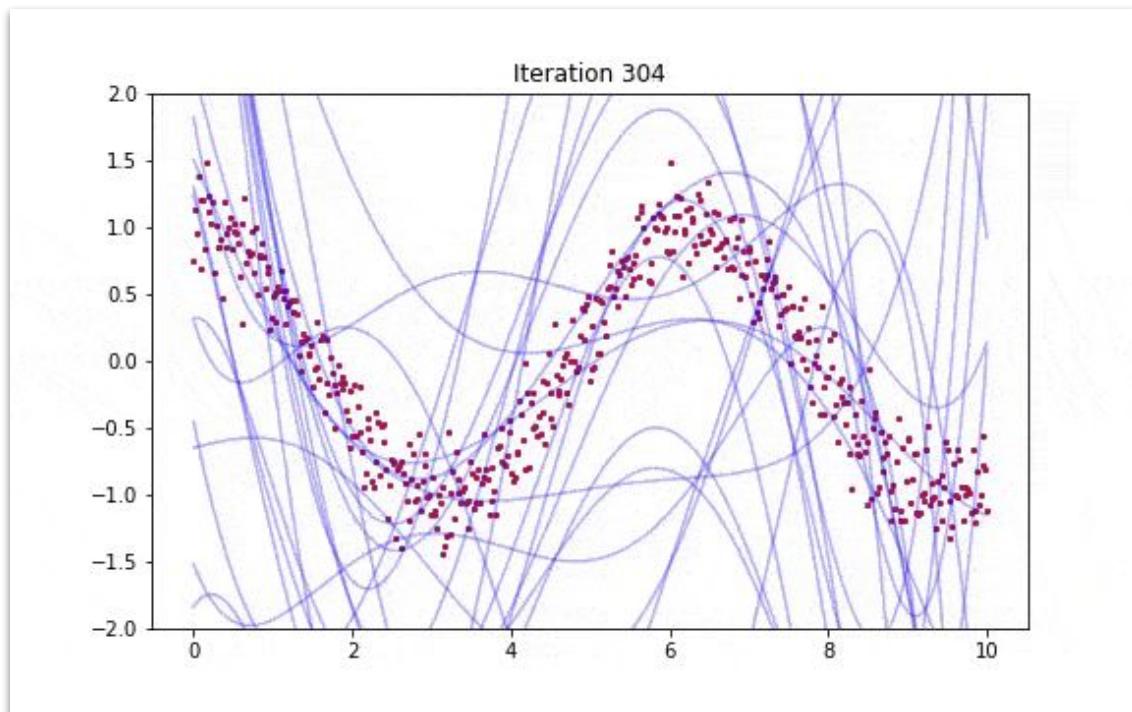
# EVOLUCIÓN DIFERENCIAL

*Differential evolution (DEA)*

- Vectores **solución** formados por **números reales**.
- **Optimización** iterativa en base a una determinada **métrica de calidad**.
- **Vectores de prueba** compiten con los individuos de la **población actual** buscando su supervivencia.
  - ↳ Se obtienen **soluciones mutadas** a partir de la diferencia de dos o más vectores de la población actual.

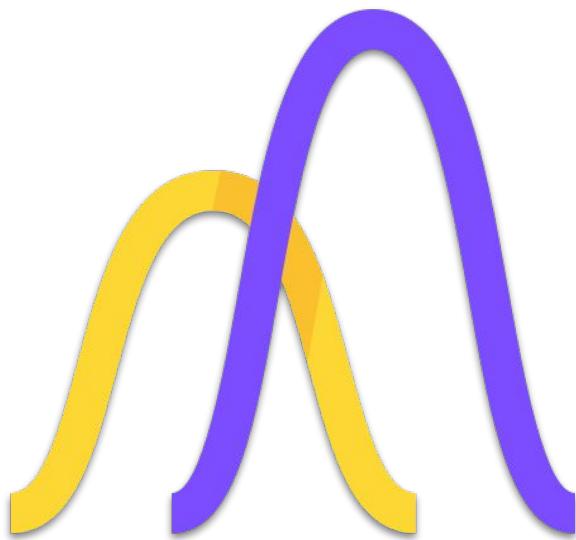


# EVOLUCIÓN DIFERENCIAL



# ESTIMACIÓN DE LA DISTRIBUCIÓN

*Estimation of distribution (EDA)*



- Estimación evolutiva de la distribución de probabilidad de cada variable (en lugar de encontrar el valor directo de las variables).
- No se emplea cruce ni mutación: la población se actualiza modificando las distribuciones de probabilidad obtenidas.
- De forma iterativa, la distribución se concentra en torno a la solución óptima.

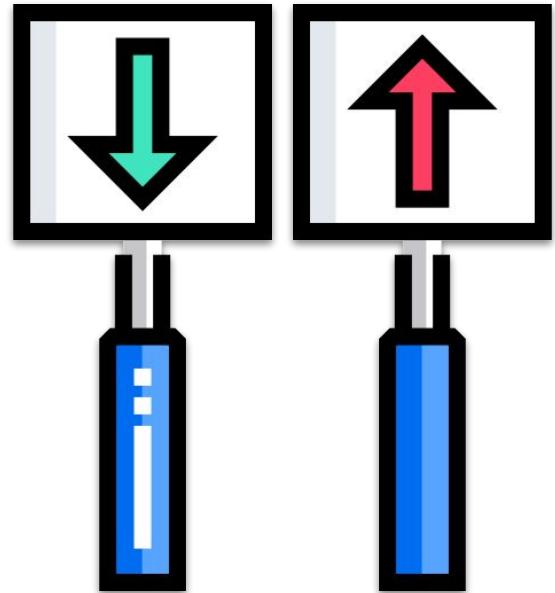
# PAES

*Pareto Archived Evolution Strategy*

Optimización multiobjetivo basada en búsqueda evolutiva.

**Problema multiobjetivo:** se pretende encontrar una o más soluciones óptimas bajo una toma de decisiones multicriterio.

Generación de soluciones en un **conjunto óptimo de Pareto**.



# Ejemplo de problema multiobjetivo

*¿Cuántos vehículos deben conformar la flota de una empresa de transporte?*

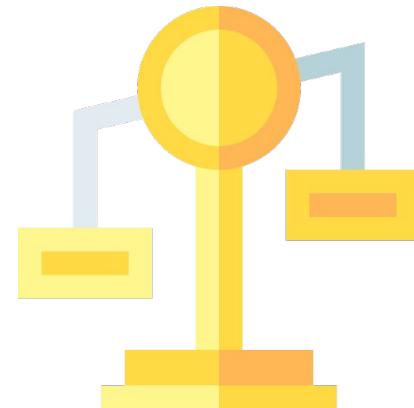


**MAXIMIZAR** → beneficio, crecimiento

**MINIMIZAR** → gastos (combustible,  
reparaciones...)

# Óptimo de Pareto

Una solución es **Pareto-óptima** cuando no existe otra solución que mejore un **objetivo** sin empeorar al menos uno de los otros.



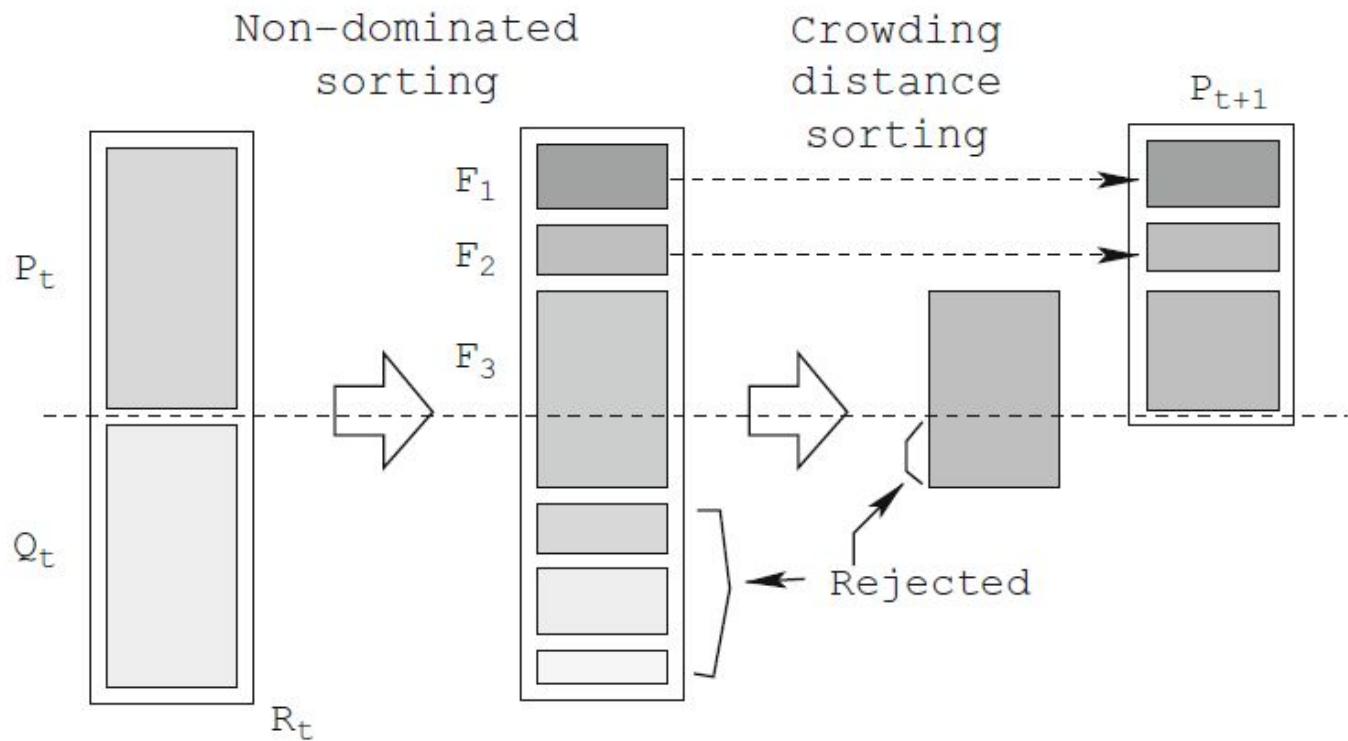
Decimos que una solución **DOMINA** a otra si es mejor.

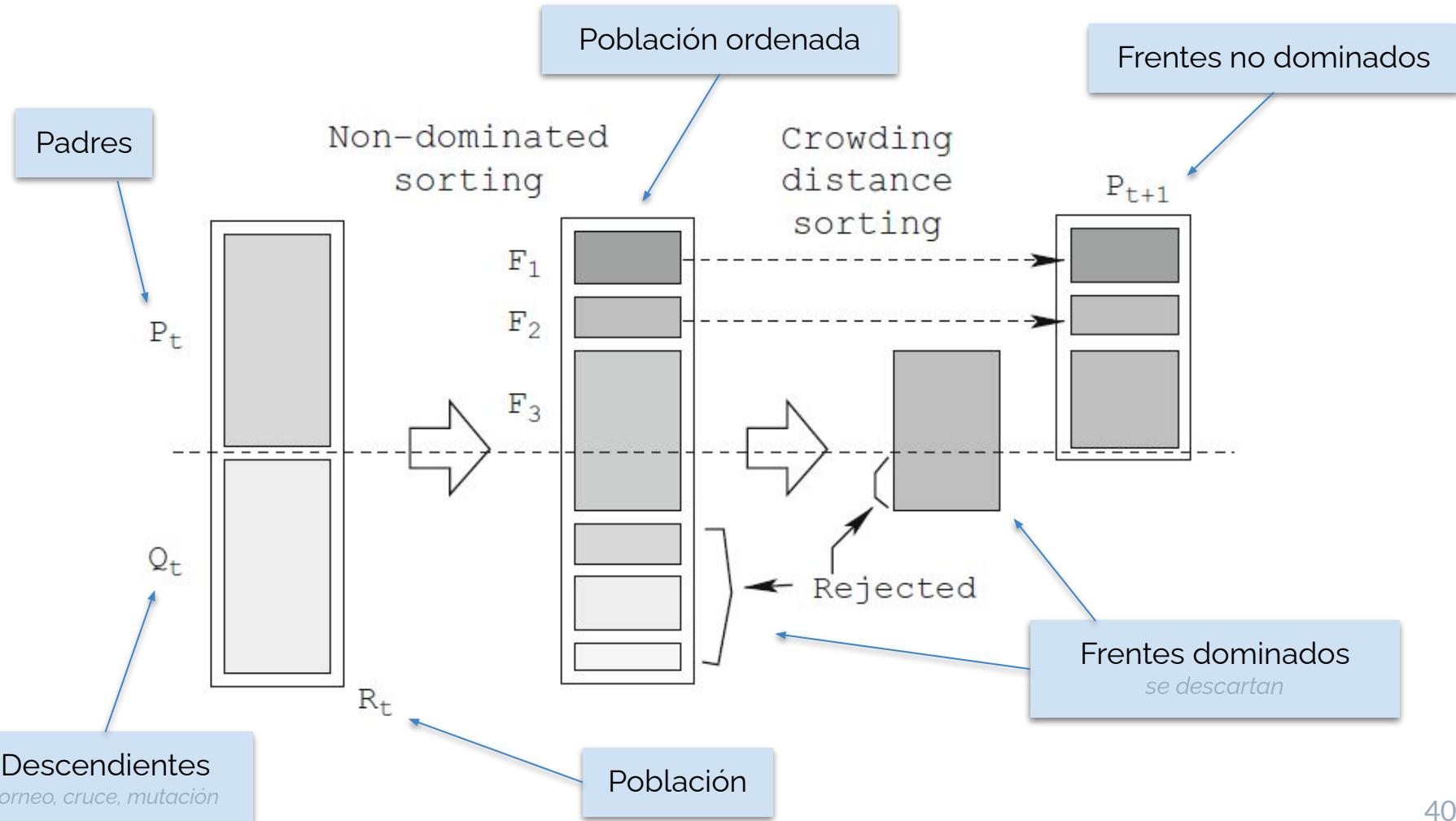
# NSGA-II

*Nondominated Sorting Genetic Algorithm*

- Algoritmo evolutivo, elitista y multiobjetivo.
- Permite la preservación y evolución de soluciones Pareto-óptimas.
- Está basado en la ordenación de la población empleando fronteras de no-dominancia.



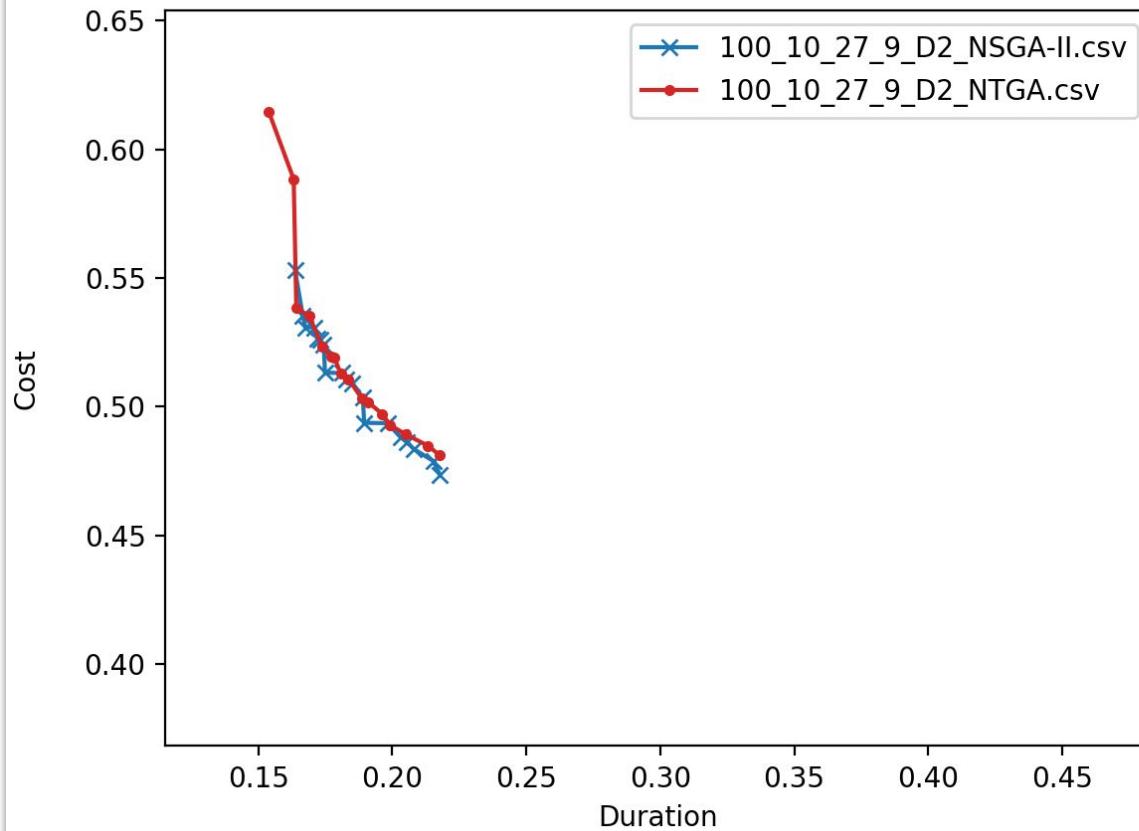






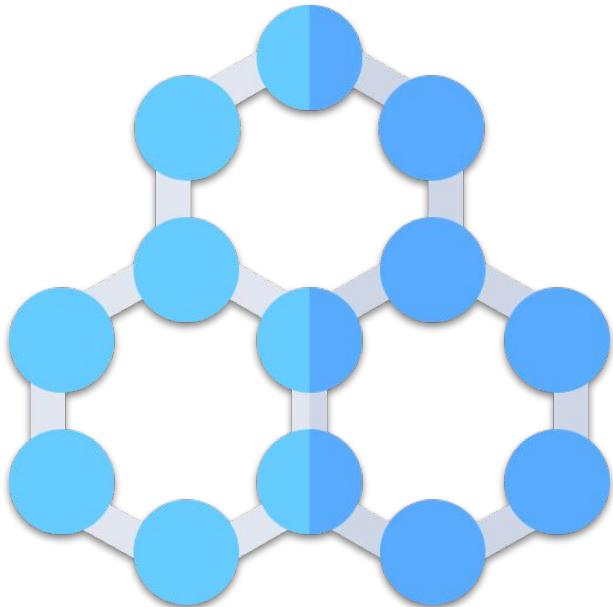
2.00

## Visualization



# ENJAMBRE DE PARTÍCULAS

*Particle Swarm Optimization (PSO)*



- Basado en el comportamiento de las **partículas** en la naturaleza y la **inteligencia de enjambre**.
  - ↳ *Bancos de peces, bandadas de aves, etc.*
- Permite optimizar un **problema** partiendo de un conjunto de **soluciones candidatas** (**partículas**) que se mueven por todo el **espacio de búsqueda**.

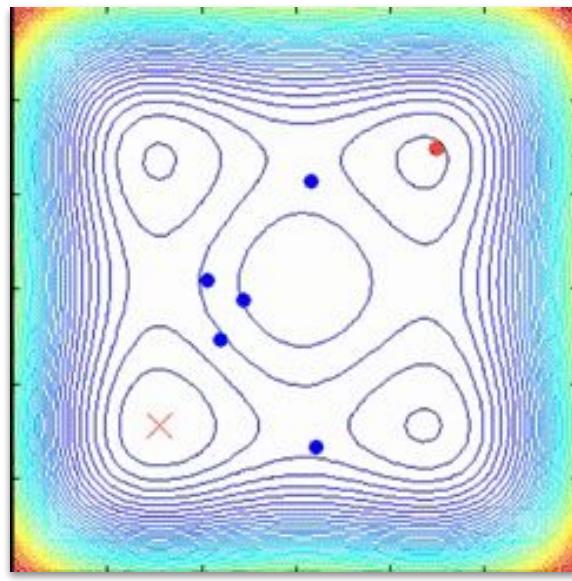
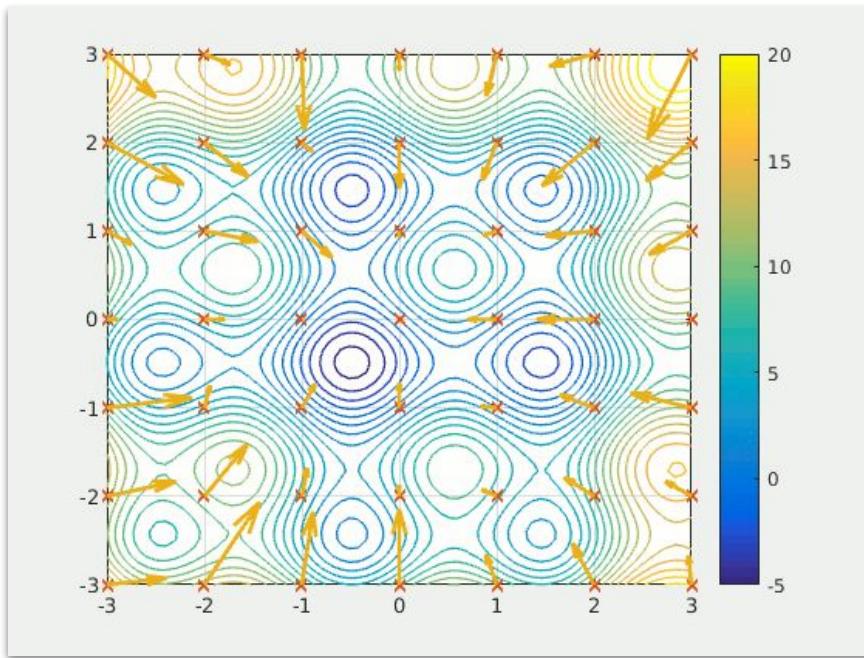
# ENJAMBRE DE PARTÍCULAS

La **posición** y la **velocidad** de las partículas determinarán cómo se produce el **movimiento** de las **partículas** por todo el **espacio** de búsqueda.

El **movimiento** de las partículas dependerá de la **mejor posición local** encontrada por cada partícula, y por las **mejores posiciones globales** del resto del enjambre.



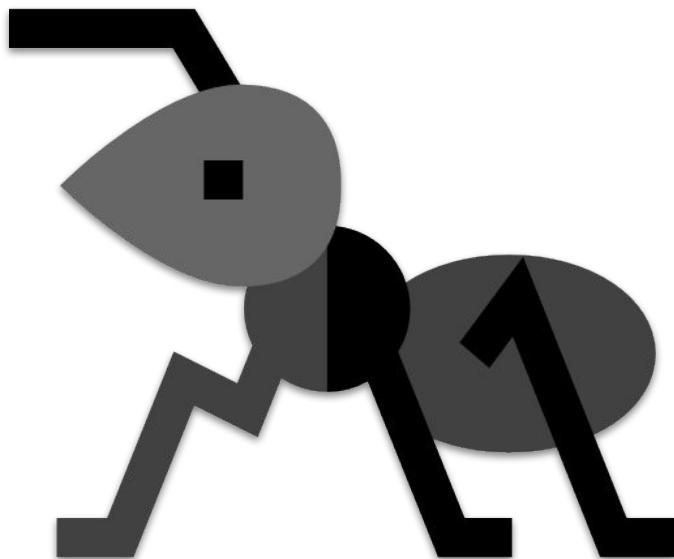
# ENJAMBRE DE PARTÍCULAS



# COLONIA DE HORMIGAS

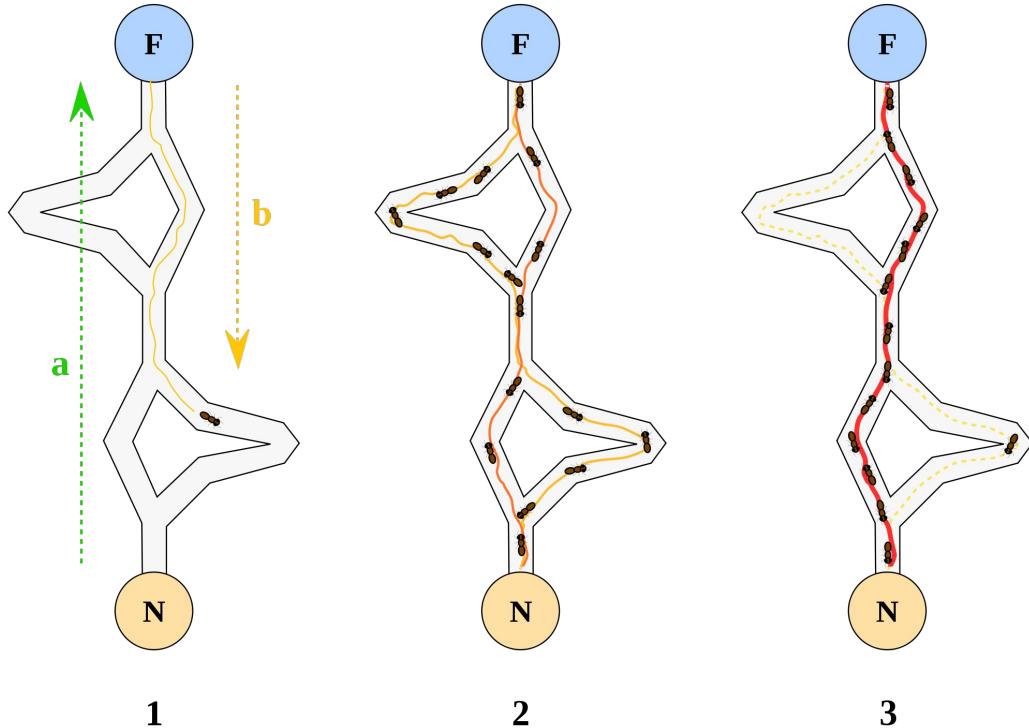
*Ant Colony Optimization (ACO)*

- Obtención del **camino óptimo** en un **grafo** basándose en el comportamiento de las **hormigas** a la hora de buscar y llevar **alimento** a la **colonia**.
- Dicho comportamiento está basado en la liberación, seguimiento y evaporación de rastros de **feromonas** por parte de las hormigas.

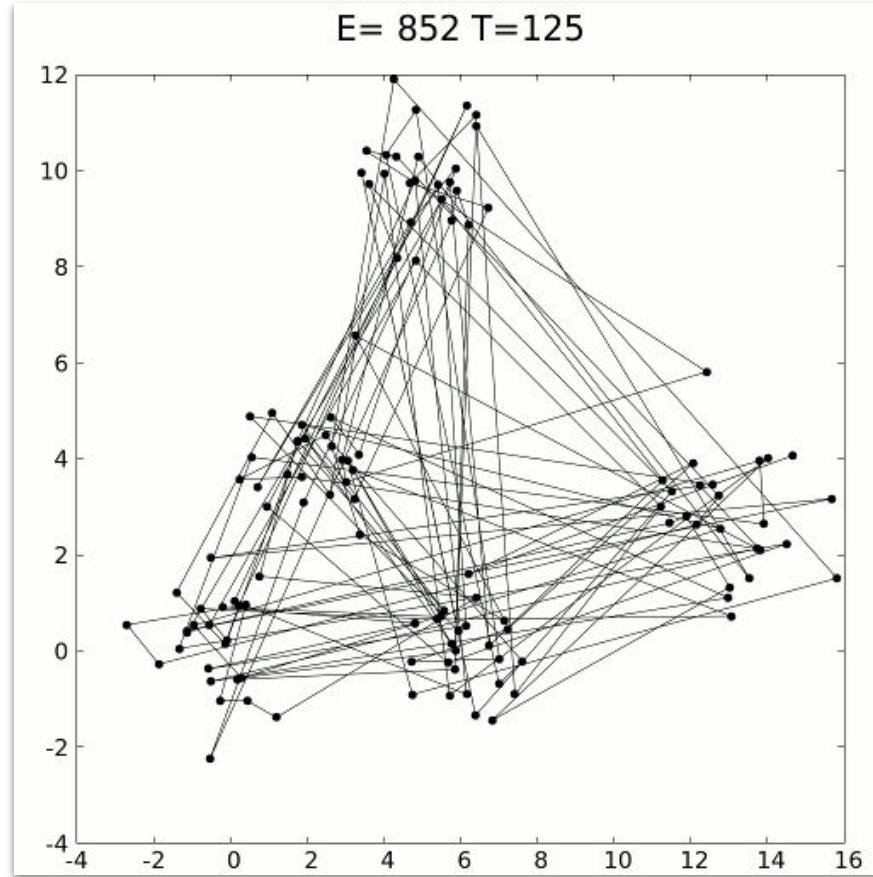


1. Una hormiga sigue un **camino aleatorio** hasta encontrar un rastro de feromonas o **alimento**.
2. En caso de encontrar un **rastro existente**, lo sigue y, si conduce a **alimento**, **lo refuerza** en el camino de vuelta a la **colonia**.
3. Por tanto, encontrar un buen camino entre la **colonia** y el **alimento** da paso a que otras hormigas sigan el mismo **camino**.

Los rastros de **feromonas** se **evaporan** con el tiempo, lo que evita la convergencia en **óptimos locales**.



La **idea** subyacente a este algoritmo es simular el comportamiento de las **hormigas** en el contexto de un **grafo** que representa el **problema** a resolver.



# Conclusiones

- Los algoritmos **bioinspirados** nos permiten resolver/aproximar **problemas** de gran complejidad.
- **inspyred** es una librería que abstrae el concepto de algoritmo **bioinspirado**, simplificando su programación y aportando una gran **flexibilidad**.

# Referencias

-  <https://pythonhosted.org/inspyred/>
-  <https://www.smithsonianmag.com/science-nature/why-knapsack-problem-all-around-us-180974333/>
-  <https://sci2s.ugr.es/sites/default/files/files/Teaching/OtherPostGraduateCourses/MasterEstructuras/2013-OyCI-Sesion-2-%20Algoritmos%20Bioinspirados.pdf>
-  <http://paginaspersonales.deusto.es/cruz.borges/Papers/1oAPIAXXI.pdf>
-  <http://www.cs.us.es/~fsancho/?e=207>
-  <http://oklahomaanalytics.com/data-science-techniques/nsga-ii-explained/>
-  Darwish, A. (2018). *Bio-inspired computing: Algorithms review, deep analysis, and the scope of applications*. Future Computing and Informatics Journal.
-  A. Cárdenas. (2012). *Inteligencia artificial, métodos bio-inspirados: un enfoque funcional para las ciencias de la computación*.
-  Dianati et al. (2018). *An Introduction to Genetic Algorithms and Evolution Strategies*.
-  Storn, R., and Price, K. (1995). *Differential Evolution—A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces*.
-  J. Knowles and D. Corne (1999). *The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation*.
-  Correa et al. (2008). *Algoritmo multiobjetivo NSGA II aplicado al problema de la mochila*.

¡Gracias!  
¿PREGUNTAS?



Andrea Morales Garzón



@andreamorgar



andreamorgarz@gmail.com

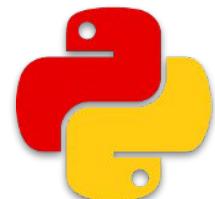
Antonio Manjavacas Lucas



@manjavacas\_



antomanjavacas@gmail.com



<https://github.com/manjavacas/algoritmos-bioinspirados>

PyConES 2021