
# Implementación de metaheurísticas en Python

# Día 2: Modelado del problema


Jesús Sánchez-Oro Calvo

Universidad
Rey Juan Carlos

# Motivación

- La calidad de los algoritmos se suele medir mediante dos métricas:
  - Valor de la **función objetivo**
  - **Tiempo** de ejecución


# ¿Cómo de importante es el programador?

- Partimos de la base de que el **algoritmo** que vamos a implementar es **bueno**
- **Si no, da igual** cómo sea el programador

# ¿Cómo de importante es el programador?

- Si el algoritmo es bueno, pero el programador **no lo es**…

# ¿Cómo de importante es el programador?

- Si el algoritmo es bueno, y el programador **es razonable**…

# ¿Cómo de importante es el programador?

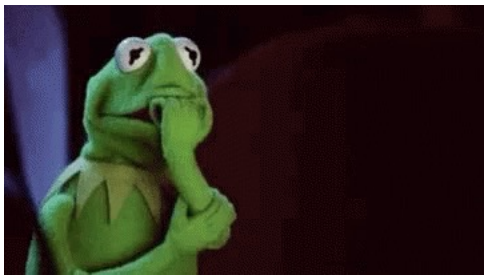
- Si tanto el algoritmo como el programador son **excelentes**…

# ¿Cómo de importante es el programador?

- ¡Cuidado al probar cosas **nuevas**!
  - Pueden salir **bien**…

  - O **mal**…

# Elección del lenguaje de programación



# ¿Cuál es el mejor lenguaje de programación?

- El mejor lenguaje de programación **no existe**
  - Si no, todos lo utilizaríamos
- ¿**Qué buscamos** en un lenguaje de programación?
  - Fácil de aprender
  - Rendimiento
  - Depuración
  - Librerías externas

# Organización del código

- Cuando nos enfrentamos a un problema de optimización, debemos pensar en la **organización** de nuestro código.
- Si el problema es similar a otros en los que ya hemos trabajado, la **estructura** será muy **parecida**.

# Organización del código

**Primera alternativa**

- La mayoría de las características que hemos implementado se repiten en casi todos los problemas.
- ¿Es realmente necesario repetir el mismo código problema tras problema?

# Organización del código

## Segunda alternativa

- Aprovechar las características del lenguaje para evitar repetir código.

# Organización del código

**Propuesta**

- Diseñar una librería con la funcionalidad básica requerida en todos los proyectos:
  - Ejecutar un algoritmo sobre un conjunto de instancias en un directorio
  - Generar una tabla con los resultados obtenidos
  - Control del tiempo de ejecución
  - …

# Organización del código

- Dividiremos el código en **paquetes** según su **funcionalidad**
  - `structure`: modelo de la instancia y de la solución
  - `constructives`: un fichero por cada constructivo
  - `localsearch`: un fichero por cada búsqueda local
  - `algorithms`: un fichero por cada algoritmo

# ¿Qué problema queremos resolver?

- **Maximización de la diversidad**
- Variante de la **suma de distancias** entre elementos seleccionados

$$f(S) = \sum_{i=1}^{|S|} \sum_{j=i+1}^{|S|} d_{ij}$$

- $d_{ij}$ es la distancia entre los elementos $i$ y $j$

# GRASP

- **Greedy Randomized Adaptive Search Procedure**
  - *Greedy*: se basa en una función voraz
  - *Randomized*: incluye cierta aleatoriedad para diversificar la búsqueda
  - *Adaptive*: la elección de los nuevos candidatos se adapta a las modificaciones de la solución en construcción
  - *Search Procedure*: se trata de un procedimiento de búsqueda
- **Dos fases**
  - Construcción
  - Búsqueda local

# Fase de construcción

1. $CL \leftarrow \{v \in V\}$
2. $v_f \leftarrow \text{Random}(CL)$
3. $S \leftarrow \{v_f\}$
4. $CL \leftarrow CL \setminus \{v_f\}$
5. **while** $CL \neq \emptyset$ **do**
6. $\quad g_{min} \leftarrow \min_{v \in CL} g(v)$
7. $\quad g_{max} \leftarrow \max_{v \in CL} g(v)$
8. $\quad \mu \leftarrow g_{max} - \alpha \cdot (g_{max} - g_{min})$
9. $\quad RCL \leftarrow \{v \in CL : g(v) \geq \mu\}$
10. $\quad v_s \leftarrow \text{Random}(RCL)$
11. $\quad S \leftarrow S \cup \{v_s\}$
12. $\quad CL \leftarrow CL \setminus \{v_s\}$
13. **endwhile**
14. **return** $S$

# Fase de construcción

1. $CL \leftarrow \{v \in V\}$
2. $v_f \leftarrow \text{Random}(CL)$
3. $S \leftarrow \{v_f\}$
4. $CL \leftarrow CL \setminus \{v_f\}$
5. **while** $CL \neq \emptyset$ **do**
6. $\quad g_{min} \leftarrow \min_{v \in CL} g(v)$
7. $\quad g_{max} \leftarrow \max_{v \in CL} g(v)$
8. $\quad \mu \leftarrow g_{max} - \alpha \cdot (g_{max} - g_{min})$
9. $\quad RCL \leftarrow \{v \in CL : g(v) \geq \mu\}$
10. $\quad v_s \leftarrow \text{Random}(RCL)$
11. $\quad S \leftarrow S \cup \{v_s\}$
12. $\quad CL \leftarrow CL \setminus \{v_s\}$
13. **endwhile**
14. **return** $S$

La **lista de candidatos** contiene todos los elementos salvo el primero, que se elige al azar.

# Fase de construcción

1. $CL \leftarrow \{v \in V\}$
2. $v_f \leftarrow \text{Random}(CL)$
3. $S \leftarrow \{v_f\}$
4. $CL \leftarrow CL \setminus \{v_f\}$
5. **while** $CL \neq \emptyset$ **do**
6. $\quad g_{min} \leftarrow \min_{v \in CL} g(v)$
7. $\quad g_{max} \leftarrow \max_{v \in CL} g(v)$
8. $\quad \mu \leftarrow g_{max} - \alpha \cdot (g_{max} - g_{min})$
9. $\quad RCL \leftarrow \{v \in CL : g(v) \geq \mu\}$
10. $\quad v_s \leftarrow \text{Random}(RCL)$
11. $\quad S \leftarrow S \cup \{v_s\}$
12. $\quad CL \leftarrow CL \setminus \{v_s\}$
13. **endwhile**
14. **return** $S$

La **lista de candidatos restringida** contiene todos los elementos que superan el umbral de calidad $\mu$.

# Fase de construcción

1. $CL \leftarrow \{v \in V\}$
2. $v_f \leftarrow \text{Random}(CL)$
3. $S \leftarrow \{v_f\}$
4. $CL \leftarrow CL \setminus \{v_f\}$
5. **while** $CL \neq \emptyset$ **do**
6. $\quad g_{min} \leftarrow \min_{v \in CL} g(v)$
7. $\quad g_{max} \leftarrow \max_{v \in CL} g(v)$
8. $\quad \mu \leftarrow g_{max} - \alpha \cdot (g_{max} - g_{min})$
9. $\quad RCL \leftarrow \{v \in CL : g(v) \geq \mu\}$
10. $\quad v_s \leftarrow \text{Random}(RCL)$
11. $\quad S \leftarrow S \cup \{v_s\}$
12. $\quad CL \leftarrow CL \setminus \{v_s\}$
13. **endwhile**
14. **return** $S$

Se elige un elemento al azar de la RCL, actualizando la lista de candidatos.

# Fase de mejora

- Implementaremos una búsqueda local sencilla
- **Movimiento**: Intercambiar un elemento que está en la solución por uno que no está

$$Move(S, u, v) \leftarrow (S \setminus \{u\}) \cup \{v\}$$

- **Vecindad**

$$N(S) \leftarrow \{S' \leftarrow Move(S, u, v)\ \forall u \in S \wedge \forall v \in V \setminus S\}$$

# Fase de mejora

## First improvement

- Se recorre la vecindad y cada vez que haya un movimiento de mejora se aplica y se comienza de nuevo la búsqueda

## Best improvement

- Se recorre la vecindad completa, aplicando el mejor movimiento que podamos encontrar

# ¡Manos a la obra!