grafo RESEARCH

Cómo implementar metaheurísticas sin morir en el intento

Jesús Sánchez-Oro

Universidad Rey Juan Carlos

Índice

Índice

Motivación

- Métricas a tener en cuenta para evaluar la **calidad** de un algoritmo:
  - Valor de la **función objetivo**
  - **Tiempo** de ejecución

¿Cómo de importante es el programador?

- Tenemos un **buen algoritmo**
- Si no, el programador no tiene nada que hacer.

¿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** …

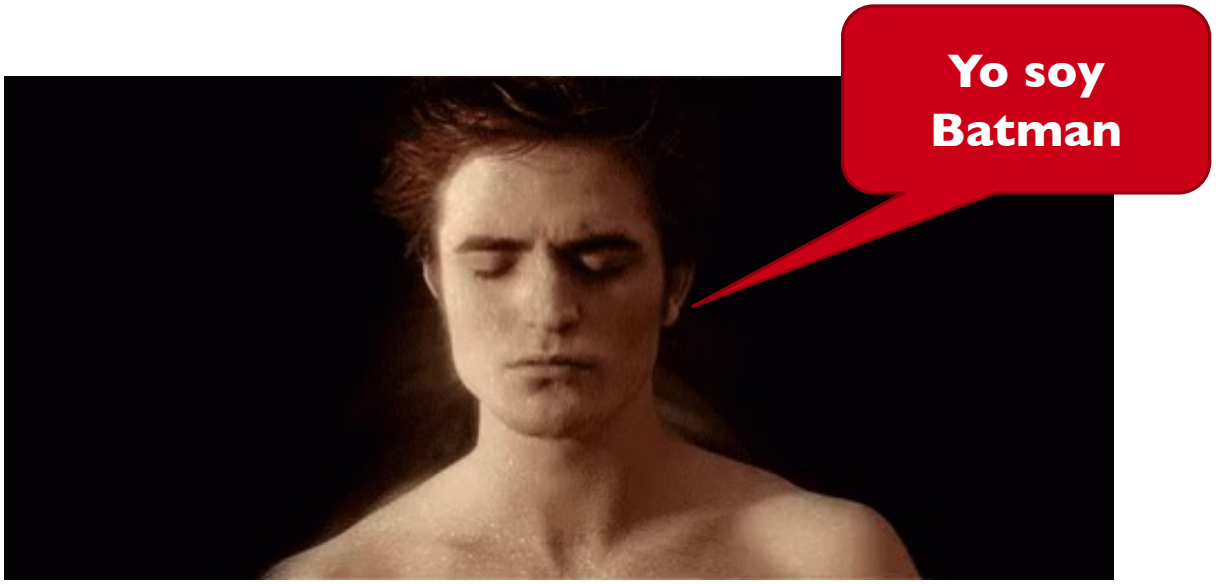
¿Cómo de importante es el programador?

- Si tanto el algoritmo como el programador son **muy buenos** …

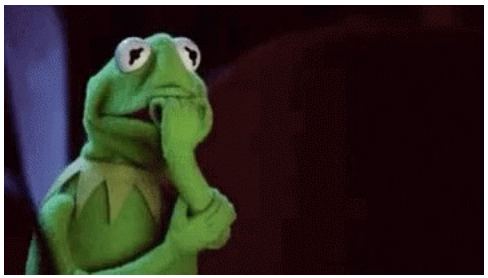
¿Cómo de importante es el programador?

- Si el programador quiere probar **cosas nuevas**

  …



Elección del lenguaje de programación



¿Cuál es el mejor lenguaje?

- El mejor lenguaje **no existe**
  - Si existiera, todos lo utilizaríamos
- ¿Qué **buscamos** en un lenguaje de programación?
  - Facilidad de aprendizaje
  - Rendimiento
  - Herramientas de depuración
  - Librerías externas

¿Por qué elegí Java?

- La **curva de aprendizaje** es bastante suave.
- La JVM se encarga de la **gestión de la memoria**.
- Paradigma **orientado a objetos**.
- Un buen código en Java **no es necesariamente más lento** que uno en C/C++.

¿Por qué elegí Java?

- El **tiempo de desarrollo** en Java es generalmente corto.
- Tiene numerosas **librerías externas** que son de gran ayuda.

Mar 2021	Mar 2020	Change	Programming Language	Ratings	Change
1	2	▲	C	15.33%	-1.00%
2	1	▼	Java	10.45%	-7.33%
3	3		Python	10.31%	+0.20%
4	4		C++	6.52%	-0.27%
5	5		C#	4.97%	-0.35%

Índice TIOBE (22/03/21)
https://www.tiobe.com/tiobe-index/

Índice

Organización de código

- Cuando nos enfrentamos a un problema nuevo, debemos invertir tiempo en pensar la **estructura de nuestro código**.
- Si el problema es similar a otro en el que hemos trabajado previamente, la **estructura será similar**.

Organización de código

- La mayoría de funcionalidades que necesitamos para un problema concreto se **repiten** para el **resto de problemas**.
- ¿Es realmente necesario **repetir el mismo código** continuamente?

Organización de código

- **Primera aproximación**
  - **Copiar y pegar** el código del último proyecto y hacer pequeñas modificaciones hasta adaptarlo.

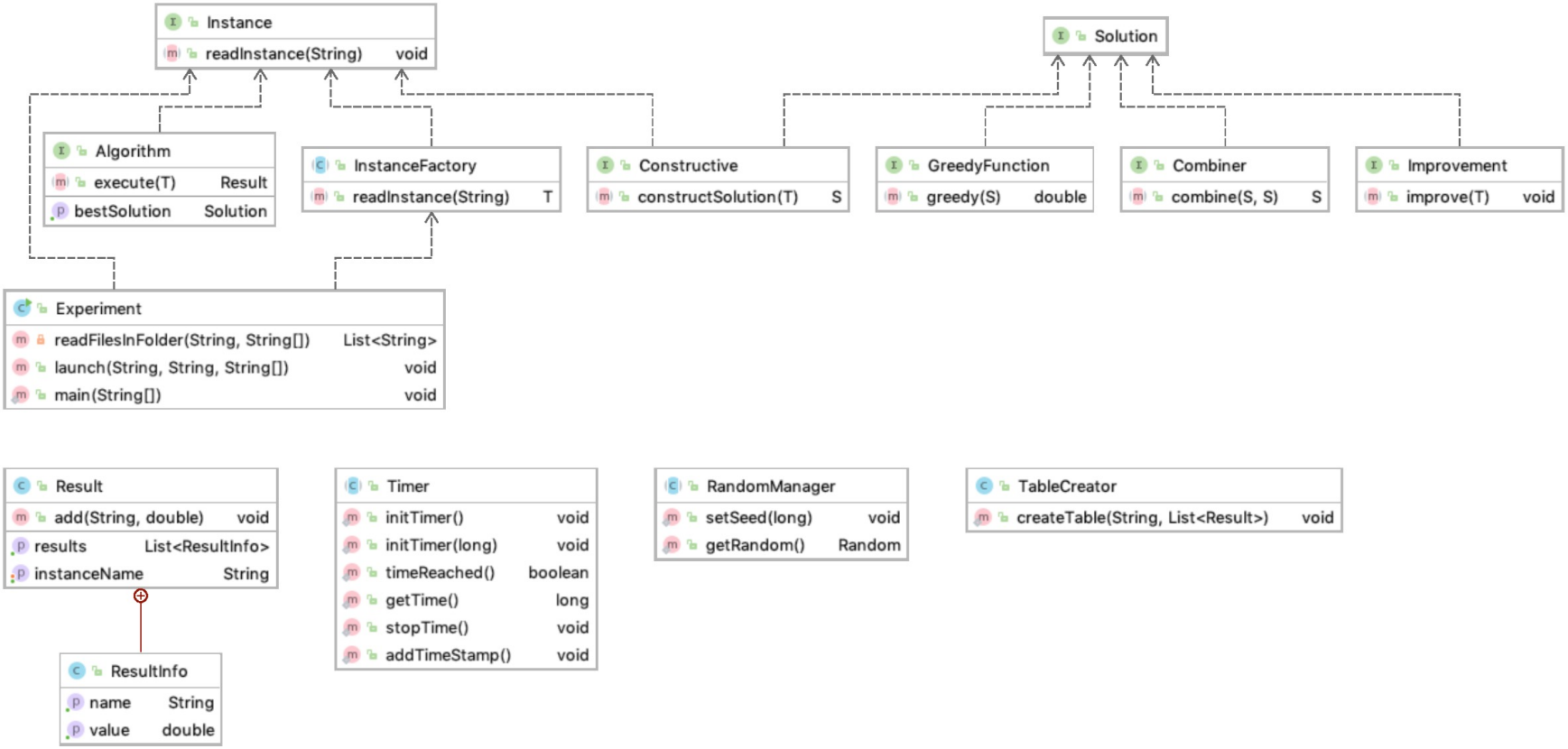
Organización de código

- **Segunda aproximación**
  - Aprovechar las **ventajas del lenguaje** para evitar repetir el código.

Organización de código

- **Propuesta**: crear una librería que contenga la funcionalidad básica que vamos a utilizar en todos los proyectos.
  - Ejecutar un algoritmo sobre un conjunto de instancias en un directorio.
  - Generar una tabla con los resultados.
  - Controlar el tiempo de cómputo.
  - …

GrafoOptiLib



Índice

Estructuras de datos

- Las ED definen **cómo se organizan los datos** de nuestro problema.
- Si elegimos la ED correcta, podremos añadir, modificar o eliminar datos de forma **eficiente**.
- Son una de las **partes clave** de nuestro código.

Estructuras de datos

- La mayoría de los lenguajes ofrecen sus **propias implementaciones** de las ED más conocidas, así que **no** solemos necesitar implementarlas.

Estructuras de datos

- Sin embargo, si necesitamos alguna ED más **compleja o específica**, tendremos que implementarla.

Estructuras de datos

- Normalmente creemos que el lenguaje de programación es la **clave para desarrollar un algoritmo eficiente**.
- Sin embargo, la clave es la **complejidad** de las estructuras de datos que utilizamos.

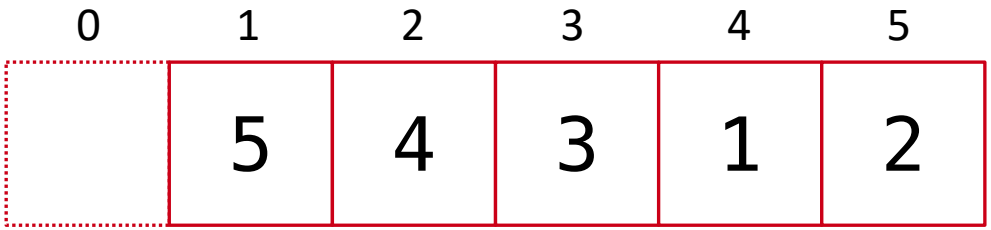
Estructuras de datos

- Si realizamos **muchas operaciones** sobre las ED, nos gustaría que fueran lo más eficiente posible.
- Debemos centrarnos en **reducir la complejidad** de las operaciones más **comunes**.
  - ¿Cuál es el coste de insertar / buscar / eliminar un elemento de una ED?

¿Es de verdad tan importante?

- Tendremos 100000 elementos almacenados en una ED.
- Prueba:
  - Búsqueda de un elementa cualquiera

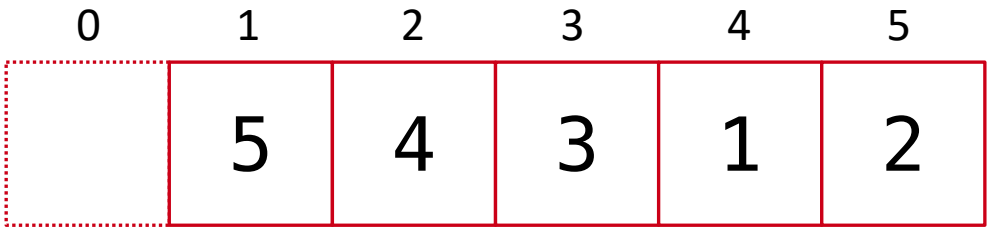
¿Por qué sucede esto?



Arrays

- Acceso a una posición en **tiempo constante**
- Mejora en la **gestión del almacenamiento**

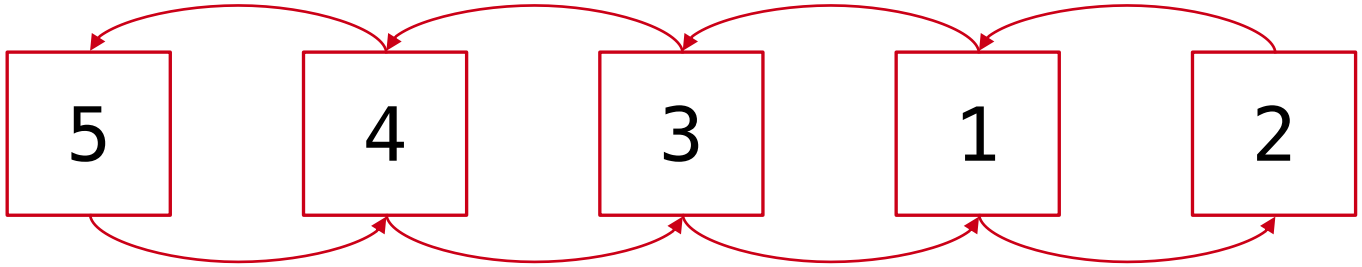
¿Por qué sucede esto?



ArrayList

- Parecidos a los arrays en representación.
- Añade más **funcionalidad** a la estructura original (como el método `contains`).

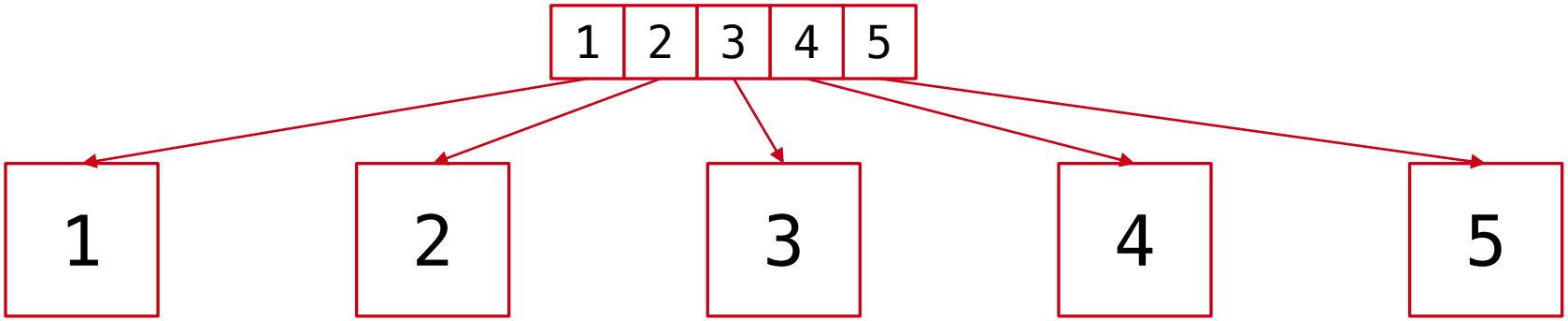
¿Por qué sucede esto?



LinkedList

- Acceso directo al **primer y último elemento** de la lista.
- Si necesitamos acceder al elemento k, tendremos que desplazarnos k posiciones desde el inicio cada vez.

¿Por qué sucede esto?



HashSet

- Cada elemento se identifica con un **número único**.
  - Necesitamos definir como asignamos ese número a cada elemento (**código hash**).
- Comprobar si un elemento está en el conjunto tiene **complejidad constante**.

¿Qué estructura de datos debería elegir?

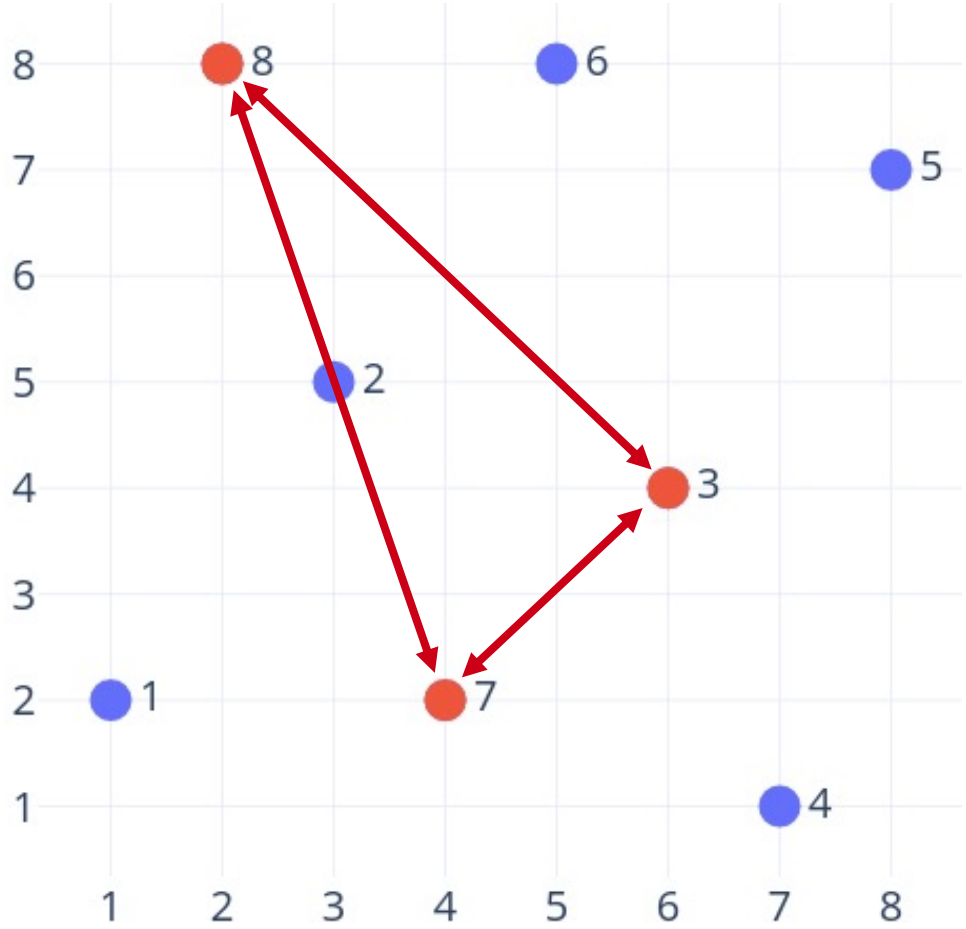
- No existe la estructura de datos **ideal**.
- Depende totalmente de las **operaciones** que realicemos con más **frecuencia**.
- **¡¡PIENSA ANTES DE PROGRAMAR!!!**

Índice

Maximum Diversity Problem (MDP)

- **Entrada**: conjunto de n elementos, distancia entre ellos, y número de elementos k que tenemos que seleccionar.
- **Objetivo**: encontrar el subconjunto de k elementos de manera que se maximiza la suma de distancias entre ellos.

Maximum Diversity Problem (MDP)



$$f(S) = d(3,8) + d(3,7) + d(7,8)$$

Maximum Diversity Problem (MDP)

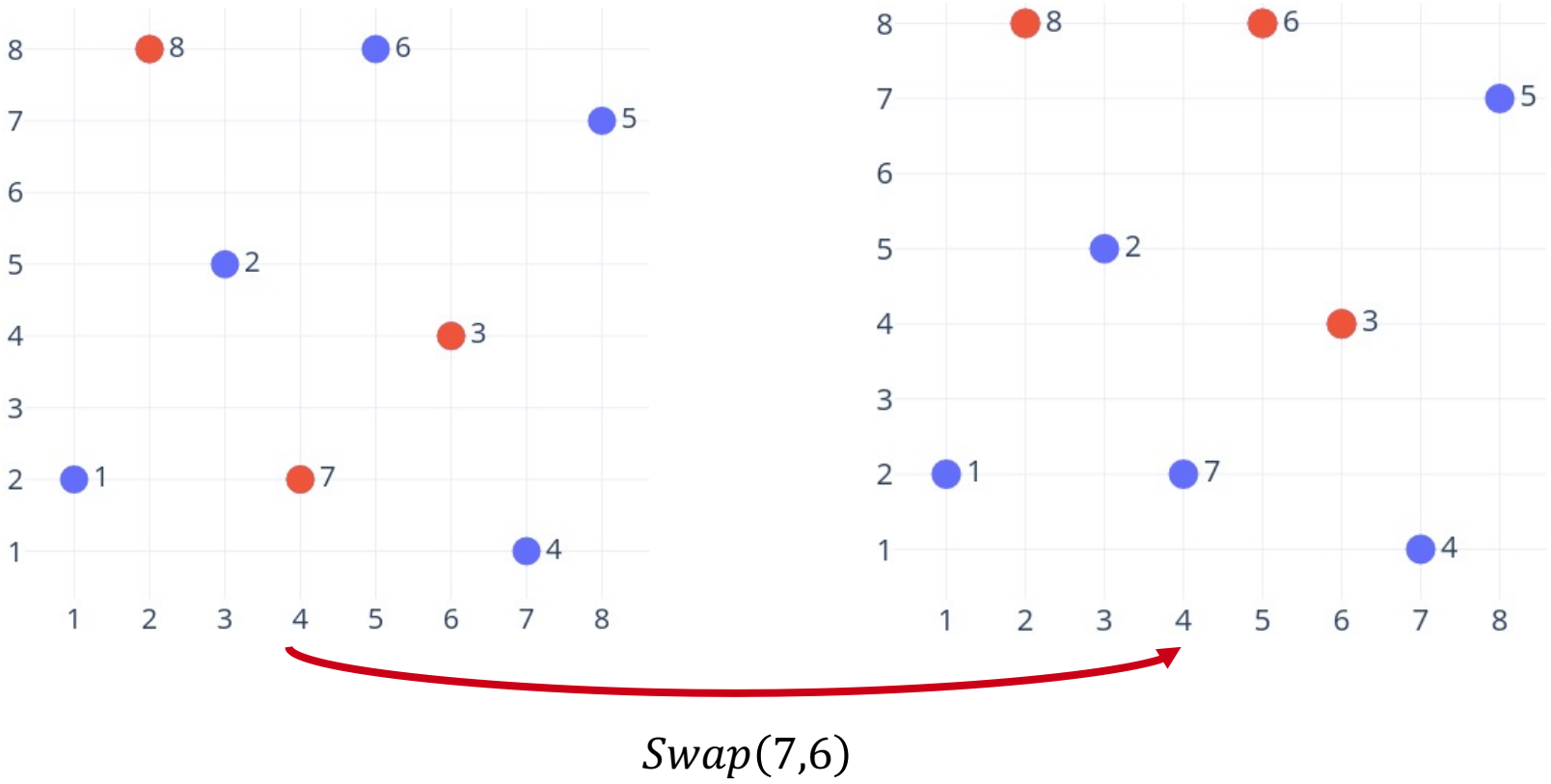
- ¿Qué conocemos de la **instancia**?
  - Número de localizaciones totales
  - Número de localizaciones que tenemos que seleccionar
  - Distancia entre ellas

Maximum Diversity Problem (MDP)

- ¿Qué conocemos de la **solución**?
  - ¿Qué instancia estamos resolviendo?
  - ¿Cuáles son los elementos que elegimos?
  - ¿Cuál es el valor de la suma de distancias entre los elegidos?

Maximum Diversity Problem (MDP)

- Movimiento básico: **intercambio**



Maximum Diversity Problem (MDP)

- **G**reedy **R**andomized **A**daptive **S**earch **P**rocedure
  - Fase de **construcción**
  - Fase de **mejora**

Maximum Diversity Problem (MDP)

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. $\quad RCL \leftarrow \{v \in CL : g(v) \leq \mu\}$
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

Maximum Diversity Problem (MDP)

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. $\quad RCL \leftarrow \{v \in CL : g(v) \leq \mu\}$
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 a todos los elementos menos al primero, que se elige de manera aleatoria.

Maximum Diversity Problem (MDP)

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. $\quad RCL \leftarrow \{v \in CL : g(v) \leq \mu\}$
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 a todos los elementos cuyo valor de una función voraz $g()$ es mejor que un cierto umbral.

Maximum Diversity Problem (MDP)

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. $\quad RCL \leftarrow \{v \in CL : g(v) \leq \mu\}$
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 al azar un valor de la **RCL**, actualizando la lista de candidatos y la solución

Maximum Diversity Problem (MDP)

Fase de mejora

- Búsqueda local basada en el movimiento de **intercambio**.
- **First improvement**
  - Se aplica el primer movimiento que produzca una mejora.
- Exploración **aleatoria** de la vecindad.

Maximum Diversity Problem (MDP)

- Vamos a probar una **implementación directa** de esta propuesta.

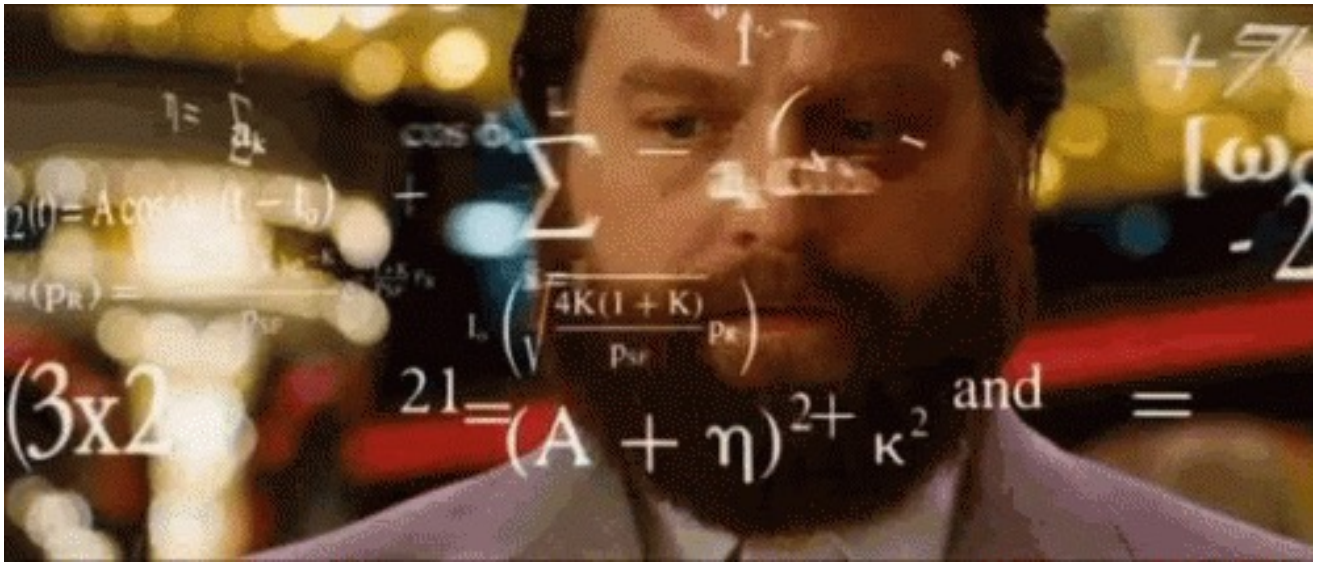
Índice

Mejoras en el código

- El código es **demasiado lento**
- Un mal rendimiento suele estar asociado a **repetir cálculos de manera innecesaria**.
- Esto es muy común en la **evaluación de la función objetivo**.

Mejoras en el código

- ¿Es realmente necesario **evaluar** el valor de la función objetivo tras cada movimiento?

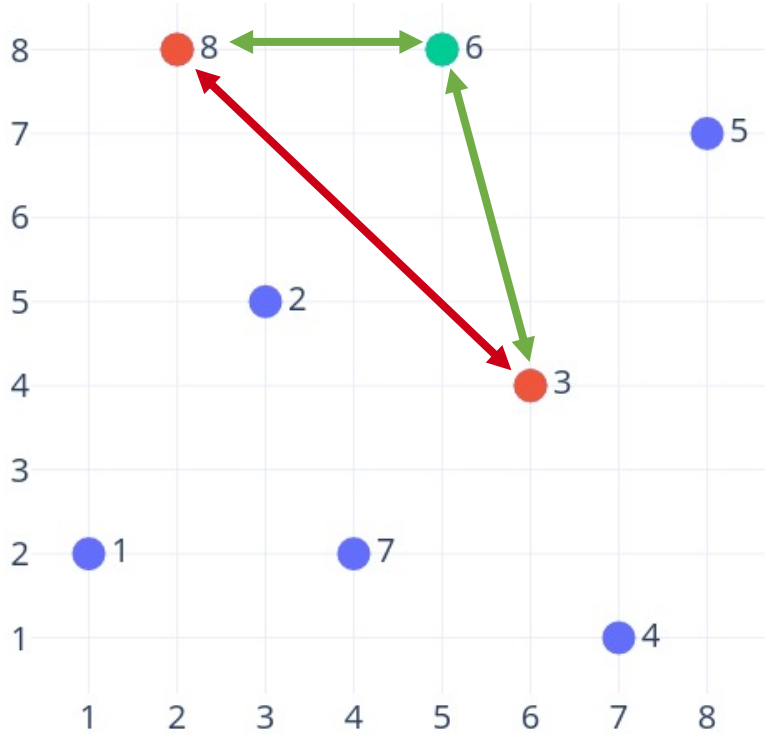


Mejoras en el código

- Tenemos que analizar qué evaluaciones son **estrictamente necesarias** para ahorrar tiempo.
- Por ejemplo, para el MDP:
  - ¿Cómo podemos actualizar el valor de la F.O. cuando **añadimos** / **eliminamos** un elemento?
  - ¿Cómo afecta a la función objetivo un **movimiento de intercambio**?

Mejoras en el código

- Cuando **añadimos** un elemento:

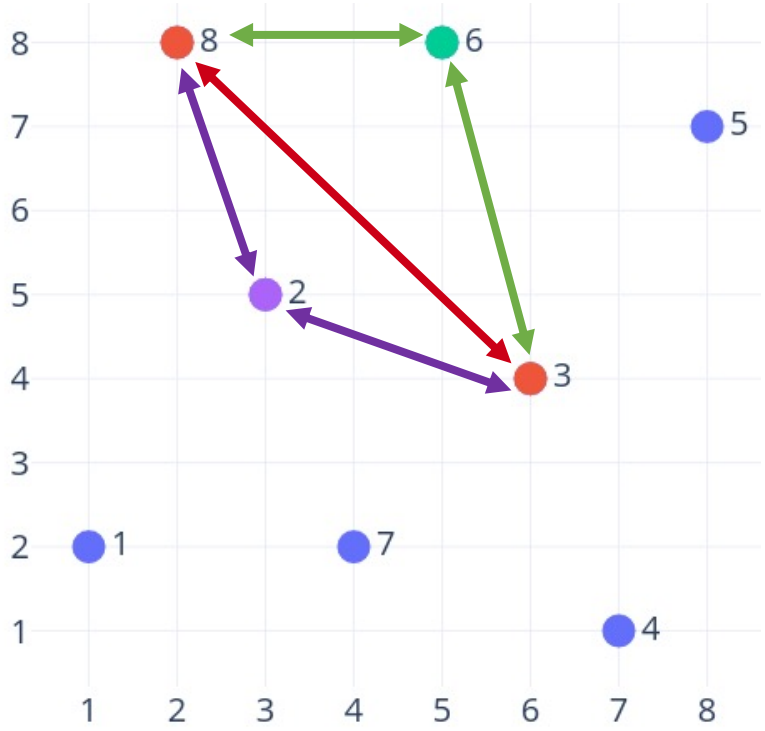


¡¡¡Nos **ahorramos** calcular de nuevo las distancias entre los elementos que ya están en la solución!!!

$$f(S \cup \{6\}) = f(S) + d(6,8) + d(3,6)$$

Mejoras en el código

- Cuando **intercambiamos** dos elementos:



$$Swap(6,2)$$

$$f(S \cup \{6\}) = f(S) - d(6,8) - d(3,6) + d(2,8) + d(2,3)$$

Mejoras en el código

- Tenemos que analizar la **complejidad** de las operaciones más comunes que vamos a realizar con nuestras **estructuras de datos**.
- Complejidad de añadir / eliminar / consultar en:
  - ArrayList
  - LinkedList
  - HashSet

Mejoras en el código

- Vamos a añadir / eliminar **muchos elementos**
  - Cualquiera de las tres ED nos sirve, tienen complejidad $O(1)$
- Vamos a **consultar** muchas veces si un elemento está o no en la solución
  - El conjunto ofrece complejidad $O(1)$
  - Las listas ofrecen complejidad $O(n)$

Mejoras en el código

- En la fase de **construcción**
  - ¿Es realmente necesario **hacer el movimiento** para evaluar la función voraz?
  - ¿Necesitamos **crear realmente la RCL** en cada paso?
- En la fase de **mejora**
  - ¿Necesitamos **realizar el movimiento** para ver si mejora y, en caso contrario, **deshacerlo**?

Índice

Paralelización

- La mayoría de nuestros ordenadores tienen **más de un núcleo**.
  - Si el tuyo no lo tiene, es momento de renovar.
- Entonces, ¿por qué tenemos que hacer **código secuencial**?

Paralelización

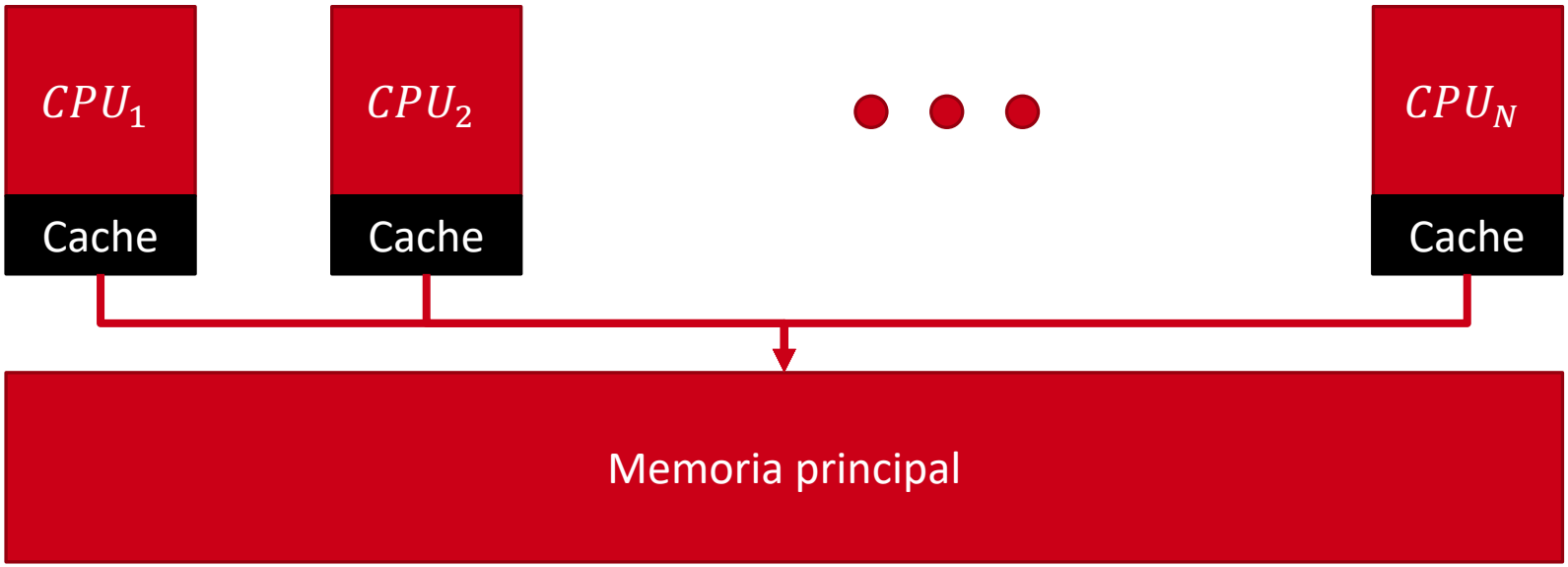
- En un problema secuencial tenemos **un solo proceso** con un **flujo de control único**.
- En un programa paralelo tenemos dos o más procesos cooperando para completar una tarea.
  - Debemos asegurar una correcta **comunicación** y **sincronización** entre procesos.

Paralelización

- **¡Con cuidado!** La potencia sin control no sirve de nada.
- Debemos aprender a paralelizar, o será **más lento que la versión secuencial**.

Modelo de memoria

- Utilizaremos un modelo de **memoria compartida**.
  - Una única memoria se comparte entre todos los procesadores.



¿Cómo puedo paralelizar código?

- Utilizando un compilador que **automáticamente** paralelice nuestro código.
- Ventajas:
  - No implica ningún trabajo extra.
- Desventajas:
  - No se puede hacer en todos los lenguajes.
  - El resultado puede no ser todo lo eficiente que creíamos.

¿Cómo puedo paralelizar código?

- Utilizando los **recursos del sistema operativo**:
  - Procesos, threads, semáforos, ficheros, etc.
- Ventajas:
  - Disponible en cualquier lenguaje de programación.
- Desventajas:
  - Ridículamente difícil en algunos casos

¿Cómo puedo paralelizar código?

- Utilizando **librerías** que faciliten la paralelización, como OpenMP.
- Ventajas:
  - Solo tenemos que modificar ligeramente nuestro código.
- Desventajas:
  - No están disponibles para todos los lenguajes.

¿Cómo puedo paralelizar código?

- Utilizando un **lenguaje de programación** preparado para el paralelismo.
- Ventajas:
  - Cualquier lenguaje moderno está preparado para ello.
- Desventajas:
  - Necesitamos modificar nuestro código en profundidad, incluso el diseño.

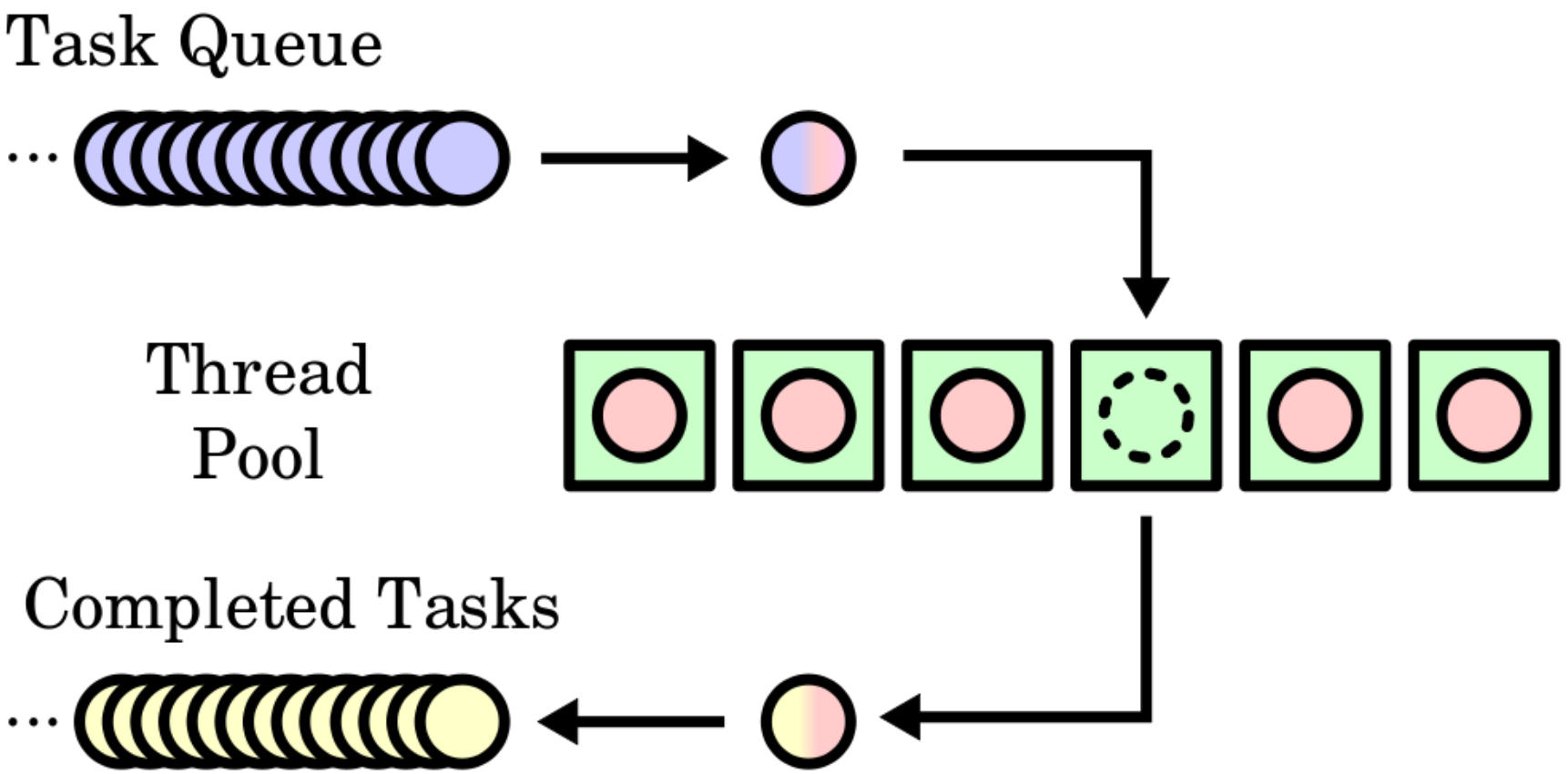
Paralelización en Java

- Java está preparado para desarrollar código paralelo de forma **rápida y sencilla**.
- Podemos utilizar herramientas de bajo nivel, pero Java ofrece **herramientas de alto nivel** para paralelizar código.

¿Cómo puedo paralelizar una metaheurística?

- Paralelizando ciertos fragmentos de código, **sin rediseñar el algoritmo**.
  - Poca contribución científica.
  - Muy sencillo.
- **Adaptando el algoritmo** para aprovechar el hardware al máximo.
  - Contribución científica relevante.
  - Más difícil.

Java Thread Pool



¿Queremos paralelizar más?

- La segunda opción implica **rediseñar el algoritmo completamente**.
- La mayoría de metaheurísticas ya tienen un **diseño paralelo** propuesto.
  - Alba, E. (2005). Parallel metaheuristics: a new class of algorithms (Vol. 47). John Wiley & Sons.
  - Crainic, T. G., & Toulouse, M. (2010). Parallel meta-heuristics. In Handbook of metaheuristics (pp. 497-541). Springer, Boston, MA.
  - Crainic, T. G. (2016). Parallel Meta-heuristic Search. Handbook of Heuristics, 1-39.

Objetivos de la paralelización

- Paralelizar **no implica necesariamente** reducir el tiempo de cómputo.
- Puede utilizarse para explorar una **región más amplia del espacio de búsqueda**.
  - Podemos guiar la búsqueda en varias direcciones simultáneamente, en lugar de seguir una única dirección.

Índice

¿Qué lenguaje debo utilizar?

- Debemos **buscar**:
  - Curva de aprendizaje suave.
  - Eficiencia.
  - Librerías externas.
  - Documentación, foros, manuales, …
  - Posibilidades de paralelización.
  - ¿Se utiliza en la comunidad de heurísticos?

¿Cómo debo organizar mi código?

- Debemos emplear tiempo en decidir la **estructura** de nuestro código.
- Si trabajamos en problemas similares, debemos plantearnos el **desarrollo de una librería** que evite repetir las tareas más tediosas.

¿Cuál es la clave para la eficiencia?

- Debemos pensar en qué **estructuras de datos** utilizar.
- La **evaluación incremental** de la función objetivo es una de las primeras optimizaciones para considerar.
- ¿Puedo utilizar una **función objetivo alternativa**?
- ¿Debería intentar un **diseño paralelo**?

Thanks!!

grafo RESEARCH

Cómo implementar metaheurísticas sin morir en el intento

Jesús Sánchez-Oro

Universidad Rey Juan Carlos