

# Memoria Práctica 1: Metaheurísticas

Integrantes del grupo: Juan Fernández Ceacero y Tomás Jesús Arellano Jiménez

Grupo: 4 - 15:30

<b>1. Definición del problema</b>	<b>2</b>
1.1. Definición	2
1.2. Parámetros de configuración	2
<b>2. Descripción de los algoritmos</b>	<b>4</b>
2.1. Algoritmo Greedy	4
2.2. Greedy Aleatorio	6
2.3. Búsqueda Local	8
2.4. Búsqueda Tabú	10
<b>3. Conclusiones</b>	<b>12</b>
Logs enlace drive:	13

# 1. Definición del problema

## 1.1. Definición

**Contexto y propósito:** Abordamos el Quadratic Assignment Problem (QAP) aplicado al diseño de una planta de fabricación/servicios de FORD. El objetivo es asignar cada departamento a exactamente una localización física de forma que se minimice el coste total de transporte/interacción, que depende del flujo entre departamentos y de la distancia entre localizaciones. El QAP es un problema NP-completo y conlleva un coste computacional elevado; por tanto, requiere metaheurísticas para obtener soluciones de alta calidad en tiempos razonables.

**Datos de entrada:** Se proporcionan dos matrices cuadradas de tamaño  $n$ :

- $F = (f_{ij})$ : flujo entre departamentos  $i$  y  $j$ .
- $D = (d_{kl})$ : distancia entre departamentos  $k$  y  $l$ .

**Representación de la solución:** Una solución se representa como una permutación  $S = \{S(1), \dots, S(n)\}$ , donde  $S(i) \in \{1, \dots, n\}$  es la localización asignada al departamento  $i$ . Esta representación satisface de forma natural las restricciones de unicidad (cada departamento ocupa una única localización y cada localización se asigna a un solo departamento) y facilita la aplicación de movimientos del vecindario basados en intercambios de posiciones.

**Función objetivo:**

El coste total de una asignación  $S$  se define como:

$$C(S) = \sum_{i=1}^n \sum_{j=1}^n f_{ij} \times d_{S(i), S(j)}$$

Intuitivamente, ubicar dos departamentos con alto flujo cerca (distancia pequeña) reduce el coste; ubicar departamentos con mucho tráfico lejos lo aumenta. Esta función se emplea en la práctica para evaluar cualquier solución candidata.

## 1.2. Parámetros de configuración

- Algoritmos: Se definen aquí los algoritmos que se quieren ejecutar, en nuestro caso Greedy, GreedyAleatorio, BusquedaLocal y BusquedaTabu.
- Dataset: Los distintos ficheros de datos con los que vamos a trabajar.
- Semillas: Las semillas usadas para la generación de soluciones en el Greedy Aleatorio.
- K: Rango de posiciones desde el principio del vector que podrá coger el Greedy Aleatorio.

- Iteraciones: Número de máximo de iteraciones de los algoritmos de búsqueda local y tabú.
- Tenencia\_tabu: Número de iteraciones que permanecerá un movimiento como tabú.
- Oscilación\_tabu: Porcentaje de que se elija un reinicio que intensifique o diversifique, en la búsqueda tabú.
- Estancamiento: Porcentaje relativo al total de iteraciones utilizado como umbral para determinar cuándo se considera que la búsqueda está estancada y debe reiniciarse, en la búsqueda tabú.

## 2. Descripción de los algoritmos

### 2.1. Algoritmo Greedy

**Descripción general:** Construye una asignación inicial emparejando los departamentos más importantes, con las localizaciones más centrales. La importancia de un departamento se calcula como la suma de su flujo hacia todas las demás y la centralidad de cada localización como la suma de sus distancias hacia todas las demás. Se ordena descendente por importancia y ascendente por centralidad, y se asigna el  $i$ -ésimo departamento más importante a la  $i$ -ésima localización más central.

#### Pseudocódigo:

##### ENTRADAS

```
F (n x n) // matriz de flujos entre departamentos
D (n x n) // matriz de distancias entre localizaciones
```

##### SALIDA

```
S // permutación: S(i) es la localización asignada al departamento i
```

##### ALGORITMO

1. Inicializar estructuras:  
    crear vector importancia[1..n] // una cifra por departamento  
    crear vector centralidad[1..n] // una cifra por localización  
    crear permutación vacía S
2. Calcular importancia de cada departamento:  
    para cada  $i$  en 1..n:  
        sumar todos los flujos entrantes y salientes de  $i$  en F  
        guardar el resultado en importancia[i]
3. Calcular centralidad de cada localización:  
    para cada  $k$  en 1..n:  
        sumar las distancias de  $k$  a todas las demás localizaciones en D  
        guardar el resultado en centralidad[k]
4. Preparar el orden de construcción:  
    U ← lista de departamentos 1..n ordenada por importancia descendente  
    L ← lista de localizaciones 1..n ordenada por centralidad ascendente
5. Construir la asignación de forma directa:  
    para  $t$  en 1..n:  
         $i \leftarrow U[t]$  // el  $t$ -ésimo más importante  
         $k \leftarrow L[t]$  // la  $t$ -ésima más central  
         $S(i) \leftarrow k$  // asignar ese departamento a esa localización
6. Devolver la solución:  
    retornar S

**Complejidad:** Calcular la importancia de cada departamento y la centralidad de cada localización cuestan  $O(n^2)$ . Las ordenaciones cuestan  $O(n \log n)$ . La construcción final es  $O(n)$ . En conjunto, el coste está dominado por  $O(n^2)$ .

	FORD01		FORD02		FORD03		FORD04	
GREEDY	Tamaño	20	Tamaño	20	Tamaño	30	Tamaño	30
	Minimo global	3542	Minimo global	26876	Minimo global	13292	Minimo global	150528
	Sol	Time	Sol	Time	Sol	Time	Sol	Time
	Ejecución	3876	0,33	32748	0.37	13768	0,71	188018
Media	9,43%	0,33	21,85%	0.37	3,58%	0,71	24,91%	0,66

El tiempo se mide en milisegundos.

**Explicación de las soluciones:** Podemos observar que el Greedy ofrece una calidad muy desigual según la instancia de datos. En FORD03 se acerca mucho al mínimo global (3,58%), en FORD01 el resultado es aceptable aunque ya se nota la brecha (9,43%), y en FORD02 y FORD04 se queda claramente corto (21,85% y 24,91%). Esto encaja con su lógica, emparejar lo más importante con lo más central y funciona cuando los flujos y distancias empujan en la misma dirección (caso FORD03), pero pierde eficacia cuando los datos no acompañan o son más complejos (FORD02 y FORD04).

## 2.2. Greedy Aleatorio

**Descripción general:** El Greedy Aleatorio es una variante del algoritmo Greedy clásico que incorpora un componente estocástico para aumentar la diversidad de las soluciones iniciales. En lugar de asignar siempre los departamentos y localizaciones según el mejor criterio determinista, el método utiliza una lista restringida de candidatos (RCL) de tamaño  $K$ , formada por los mejores elementos según su importancia o centralidad. A partir de esta lista, y utilizando una semilla aleatoria, el algoritmo selecciona de forma aleatoria uno de los mejores candidatos en cada paso. De esta manera, se generan soluciones factibles distintas en cada ejecución, evitando la excesiva codicia del método original y proporcionando puntos de partida variados para los algoritmos de mejora posteriores.

### Pseudocódigo:

```
ENTRADAS
  F (n x n)    // matriz de flujos entre departamentos
  D (n x n)    // matriz de distancias entre localizaciones
  k ≥ 1        // tamaño de la lista restringida de candidatos (RCL)
  σ            // semilla para el generador aleatorio

SALIDA
  S            // permutación: S(i) es la localización asignada al departamento i

ALGORITMO
  1. Inicializar estructuras:
    crear vectores importancia[1..n] y centralidad[1..n]
    crear permutación vacía S
    inicializar el generador de números aleatorios con la semilla σ

  2. Calcular importancia de cada departamento:
    para cada i en 1..n:
      importancia[i] ← suma de todos los flujos entrantes y salientes de i en F

  3. Calcular centralidad de cada localización:
    para cada k_loc en 1..n:
      centralidad[k_loc] ← suma de distancias de k_loc a todas las demás en D

  4. Ordenar:
    U ← departamentos ordenados por importancia descendente
    L ← localizaciones ordenadas por centralidad ascendente

  5. Asignar de forma aleatoria dentro del rango RCL:
    mientras U y L no estén vacías:
      rDep ← min(k, |U|); rLoc ← min(k, |L|)
      posDep ← número aleatorio en {1..rDep}; posLoc ← número aleatorio en {1..rLoc}
      i ← U[posDep]; l ← L[posLoc]
      S(i) ← l
      eliminar U[posDep] y L[posLoc]

  6. Devolver la solución:
    retornar S
```

**Complejidad:** Calcular la importancia de cada departamento y la centralidad de cada localización cuesta  $O(n^2)$ . Las ordenaciones de departamentos y localizaciones cuestan  $O(n \log n)$ . Durante la construcción, las selecciones aleatorias dentro del rango  $k$  y las asignaciones tienen coste  $O(n)$ . En conjunto, el coste total está dominado por  $O(n^2)$ , igual que en el Greedy clásico.

	FORD01		FORD02		FORD03		FORD04	
GREEDY ALEATORIO $O(n^2)$	Tamaño	20	Tamaño	20	Tamaño	30	Tamaño	30
	Minimo global	3542	Minimo global	26876	Minimo global	13292	Minimo global	150528
	Sol	Time	Sol	Time	Sol	Time	Sol	Time
Ejecución 1	3874,00	3,32	35128,00	0,65	13748,00	2,23	193432,00	3,12
Ejecución 2	3862,00	1,46	34982,00	0,89	13806,00	1,34	194068,00	3,45
Ejecución 3	3832,00	1,47	33646,00	1,21	13748,00	1,45	192118,00	2,40
Ejecución 4	3898,00	1,23	34874,00	1,12	13806,00	1,65	191562,00	2,11
Ejecución 5	3866,00	1,90	35742,00	1,11	13800,00	2,12	190834,00	1,87
Media	9,16%	1,88	29,76%	1,00	3,68%	1,76	27,82%	2,59
Desv. típica	0,01	0,84	0,03	0,23	0,00	0,40	0,01	0,67

El tiempo se mide en milisegundos.

**Explicación de las soluciones:** Observemos que las cinco ejecuciones muestran que el método aporta algo de diversidad pero su calidad sigue dependiendo mucho de la instancia. Por ejemplo, en FORD03 se comporta muy bien y queda cerca del mínimo global (3,68% de desviación media), en FORD01 ofrece un resultado aceptable aunque con margen (9,16%), y en FORD02 y FORD04 se queda claramente lejos (29,76% y 27,82%), lo que sugiere que, tal como está configurado ( $k$  y semillas), la aleatoriedad no basta para resolver los casos más complicados; además, la variación entre ejecuciones es mínima, señal de que el procedimiento explora pocas alternativas efectivas. Dicho todo esto, es importante destacar de que el algoritmo genera puntos de partida razonables para futuros algoritmos (BL y BT) y a veces mejores que el Greedy puro (caso FORD01), pero para FORD02 y FORD04 conviene apoyarse en mejora posterior (búsqueda local/tabú) o subir el parámetro  $k$  para rascar mejores combinaciones.

## 2.3. Búsqueda Local

**Descripción general:** Partiendo de una solución inicial (obtenida mediante Greedy Aleatorio), el algoritmo aplica un proceso de mejora iterativa basado en el vecindario 2-opt, donde se intercambian las localizaciones asignadas a dos unidades  $i$  y  $j$ . Este mecanismo permite explorar de forma eficiente diferentes permutaciones mediante intercambios de pares. Para evaluar rápidamente el efecto de cada movimiento, se emplea una evaluación incremental (delta) que calcula el cambio de coste del intercambio en  $O(n)$ . Además, se utilizan los Don't-Look Bits (DLB) para omitir unidades que no han generado mejoras en iteraciones recientes, reduciendo el número de evaluaciones y acelerando la convergencia del método.

### Pseudocódigo:

```
ENTRADAS
  F (n x n) // matriz de flujos entre departamentos
  D (n x n) // matriz de distancias entre localizaciones
   $\sigma$  // semilla para el Greedy Aleatorio
   $k \geq 1$  // tamaño RCL del Greedy Aleatorio
  Imax // límite máximo de iteraciones

SALIDA
  S* // mejor solución encontrada

ALGORITMO
  1. Generar una solución inicial S mediante GreedyAleatorio(F, D,  $\sigma$ , k)
     guardar S*  $\leftarrow$  S
     Inicializar DLB[1..n]  $\leftarrow$  0 e iteración it  $\leftarrow$  0

  3. Repetir hasta que no haya mejoras o se alcance Imax:
     mejora  $\leftarrow$  falso
     para cada departamento i en 1..n:
       si DLB[i] = 1 continuar
       para cada j en (i+1)..n:
          $\Delta \leftarrow \text{deltaSwap}(F, D, S, i, j)$ 
         si  $\Delta < 0$ : // el intercambio resulta en una mejor solución
           it  $\leftarrow$  it + 1
           aplicar swap(S, i, j) // 2-opt
           S*  $\leftarrow$  S
           DLB[i]  $\leftarrow$  0; DLB[j]  $\leftarrow$  0; mejora  $\leftarrow$  verdadero
       si no hubo mejora para i entonces DLB[i]  $\leftarrow$  1

  4. Devolver S*
```

Cabe destacar que durante la ejecución del algoritmo, en ningún caso calculamos el coste de las soluciones que se van generando. Realmente lo que calculamos es si el coste mejora al hacer el 2-opt. Esto lo hacemos con el deltaSwap (utilizamos las funciones de las transparencias en platea).

**Complejidad:** En el peor caso, la búsqueda local evalúa  $O(n^2)$  intercambios, cada uno con un coste  $O(n)$ , dando un total de  $O(n^3)$ . Gracias al uso de DLB, el número real de evaluaciones es menor, por lo que en la práctica el tiempo de ejecución suele ser inferior al teórico.



	FORD01		FORD02		FORD03		FORD04	
BL	Tamaño	20	Tamaño	20	Tamaño	30	Tamaño	30
	Minimo global	3542	Minimo global	26876	Minimo global	13292	Minimo global	150528
	<i>Sol</i>	<i>Time</i>	<i>Sol</i>	<i>Time</i>	<i>Sol</i>	<i>Time</i>	<i>Sol</i>	<i>Time</i>
Ejecución 1	3652,00	47,55	31472,00	20,23	13376,00	19,23	175336,00	14,23
Ejecución 2	3642,00	45,23	31736,00	21,33	13404,00	24,12	176674,00	15,22
Ejecución 3	3686,00	46,12	31402,00	21,43	13374,00	19,33	178590,00	16,41
Ejecución 4	3658,00	30,34	30730,00	25,78	13418,00	20,98	178308,00	28,23
Ejecución 5	3666,00	31,45	30674,00	23,67	13382,00	18,23	175820,00	24,22
Media	3,35%	40,14	16,10%	22,49	0,74%	20,38	17,55%	19,66
Desv. típica	0,00	8,49	0,02	2,22	0,00	2,31	0,01	6,20

El tiempo se mide en milisegundos.

**Explicación de las soluciones:** La Búsqueda Local mejora de forma clara las soluciones de partida y, en varias instancias de datos, las deja muy cerca del mínimo global. En FORD03 la brecha cae hasta un 0,74% (prácticamente óptima) y en FORD01 baja al 3,35% lo cual es una mejora notable frente al Greedy. En las instancias más difíciles (FORD02 y FORD04) también hay avance, reduciendo la desviación media al 16,10% y 17,55% respectivamente, aunque aún queda margen de mejora. Además, la variabilidad entre repeticiones es prácticamente nula (desviación típica ~0), señal de un comportamiento estable. Dado un buen punto de partida, la BL explota bien el vecindario y converge a soluciones consistentes, especialmente cuando la estructura de flujos y distancias favorece intercambios 2-opt efectivos.

## 2.4. Búsqueda Tabú

**Descripción general:** Partiendo de una solución inicial generada mediante el Greedy Aleatorio, la Búsqueda Tabú aplica un proceso iterativo de mejora que explora el vecindario 2-opt, basado en intercambios entre pares de departamentos, incorporando mecanismos de memoria para evitar ciclos y escapar de óptimos locales. El algoritmo mantiene una memoria a corto plazo mediante una lista tabú, donde cada movimiento reciente permanece prohibido durante un número fijo de iteraciones denominado tenencia (T). Además, utiliza un criterio de aspiración que permite aceptar un movimiento tabú si este mejora la mejor solución global obtenida hasta el momento. También se emplea una memoria a largo plazo que registra la frecuencia con la que cada departamento ocupa una localización determinada. Esta información se usa para aplicar estrategias de intensificación, que refuerzan configuraciones prometedoras, o de diversificación, que fomentan la exploración de nuevas zonas del espacio de soluciones. La transición entre ambos modos se controla mediante un parámetro de oscilación, que regula la probabilidad de cambiar de estrategia. Cuando el algoritmo detecta estancamiento, es decir, un porcentaje determinado de iteraciones sin mejora, alterna entre intensificar o diversificar para reactivar la búsqueda.

### Pseudocódigo:

#### ENTRADAS

F, D,  $\sigma$ , k, iteraciones, TENENCIA\_TABU, OSCILACION $\in(0,1)$ , ESTANCAMIENTO $\in(0,1)$

#### SALIDA

S\*: mejor permutación

#### ALGORITMO

```
1) S ← GreedyAleatorio(F, D,  $\sigma$ , k); C ← coste(S); S* ← S; C* ← C; n ← |S|
   TabuExp[n][n] ← 0; Freq[n][n] ← 0; DLB[1..n] ← 0
   para u=1..n: Freq[u][S[u]-1]++

   umbralEst ← máx(1, redondear(ESTANCAMIENTO·iteraciones))
   sinMejora ← 0; modoIntensificar ← verdadero; ventanaOsc ← 0

2) para iter=0..iteraciones-1:
   si (ventanaOsc=0) y (sinMejora ≥ umbralEst):
     modoIntensificar ← (aleatorio())<OSCILACION
     ventanaOsc ← umbralEst

   mejorI←-1; mejorJ←-1; mejorΔ←+∞; mejorScore←+∞; existe=falso

   para i=0..n-1:
     si DLB[i]=1 continuar
     para j=i+1..n-1:
       Δ ← deltaSwap(F,D,S,i,j) // O(n)
       tabu ← (TabuExp[i][j] > iter)
       aspira ← tabu y (C+Δ < C*)
       si tabu y no aspira continuar
       score ← Δ + ajusteOsc(Freq,S,i,j,modoIntensificar,OSCILACION,iter)
       si (score<mejorScore) o (score= y Δ<mejorΔ):
         mejorScore=score; mejorΔ=Δ; mejorI=i; mejorJ=j; existe=verdadero
     si no hubo candidato para i: DLB[i]←1
```

```

si (no existe) o (todos DLB=1):
    mejorI=-1; mejorJ=-1; mejorΔ=+∞; mejorScore=-∞
    para i=0..n-1:
        para j=i+1..n-1:
            Δ, tabu, aspira como antes; si tabu y no aspira continuar
            score como antes; actualizar mejor si procede
        si mejorI=-1: salir
    poner DLB[1..n]=0

swap(S,mejorI,mejorJ); C=C+mejorΔ
TabuExp[mejorI][mejorJ] ← iter + TENENCIA_TABU
DLB[mejorI]=0; DLB[mejorJ]=0
Freq[mejorI][S[mejorI]-1]++; Freq[mejorJ][S[mejorJ]-1]++

si C<C*: S←copia(S); C←C; sinMejora=0
si no:    sinMejora++

si ventana0sc>0: ventana0sc--

3) devolver S*

```

**Complejidad:** En cada iteración se evalúan aproximadamente  $n^2$  posibles intercambios (vecindario 2-opt). Cada evaluación deltaSwap se calcula en  $O(n)$ , por lo que el coste de una iteración es  $O(n^3)$  en el peor caso. El uso de la lista tabú y la gestión de memorias añade solo un coste constante adicional por movimiento. En la práctica, el número de intercambios evaluados suele ser menor gracias a las restricciones tabú y a los mecanismos de intensificación y diversificación, por lo que el tiempo real de ejecución es inferior al coste teórico  $O(n^3)$ .

	FORD01		FORD02		FORD03		FORD04	
BT	Tamaño	20	Tamaño	20	Tamaño	30	Tamaño	30
	Minimo global	3542	Minimo global	26876	Minimo global	13292	Minimo global	150528
	Sol	Time	Sol	Time	Sol	Time	Sol	Time
Ejecución 1	3602,00	123,43	26876,00	0,00	13350,00	304,23	150528,00	312,34
Ejecución 2	3586,00	111,76	26876,00	0,00	13380,00	312,22	173322,00	325,65
Ejecución 3	3542,00	145,13	26876,00	0,00	13360,00	298,98	150528,00	321,22
Ejecución 4	3578,00	99,10	26876,00	0,00	13384,00	287,45	173302,00	312,60
Ejecución 5	3588,00	102,67	30818,00	0,00	13372,00	349,23	150528,00	356,12
Media	1,05%	116,42	2,93%	0,00	0,58%	310,42	6,05%	325,59
Desv. típica	0,01	18,60	0,07	0,00	0,00	23,49	0,08	18,00

El tiempo se mide en milisegundos.

**Explicación de las soluciones:** La Búsqueda Tabú es claramente la más eficaz. En FORD03 deja la solución, de media, a un 0,58% del mínimo global (prácticamente óptima y muy estable) y en FORD01 al 1,05%. En FORD02 alcanza el óptimo en cuatro de cinco ejecuciones y la media (2,93%) queda penalizada por una única ejecución. En FORD04 conviven ejecuciones óptimas con otras alrededor del 15% de desviación, lo que da una media del 6,05% y refleja la alternancia entre intensificación y diversificación propia del método. En conjunto, la memoria tabú y la oscilación permiten salir de óptimos locales y acercarse mucho al mínimo global en la mayoría de las instancias, superando con holgura las soluciones Greedy aleatorias de las que partimos.

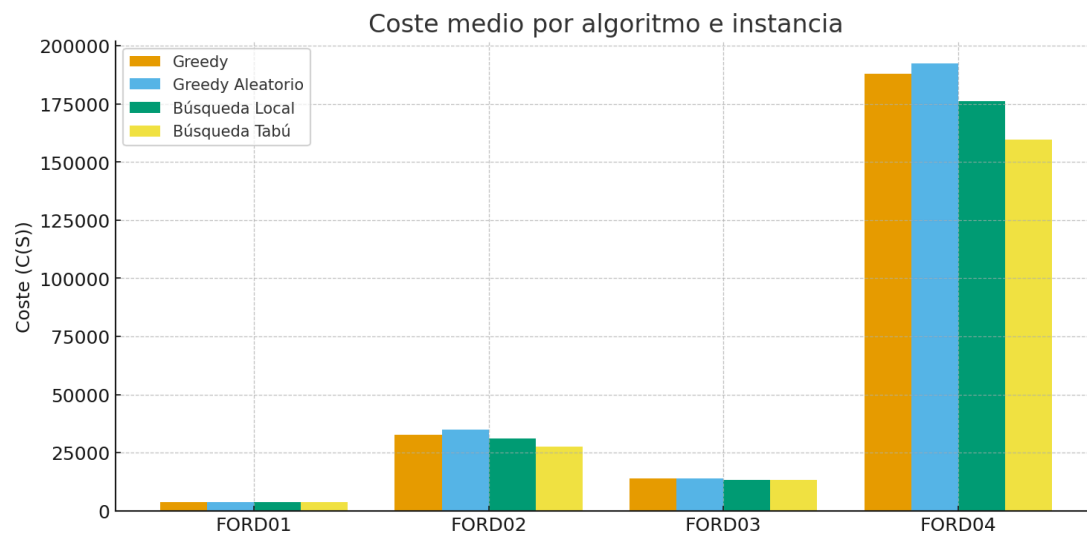
### 3. Conclusiones

El análisis de los resultados muestra claramente que existe una relación entre el tiempo que tarda cada algoritmo y la calidad de las soluciones que consigue. A partir de las pruebas realizadas, se pueden extraer varias conclusiones sobre el comportamiento de cada uno:

- El algoritmo Greedy es el más rápido, ya que construye la solución de forma directa sin realizar ningún tipo de búsqueda adicional. Su simplicidad le permite ofrecer resultados aceptables en los casos más sencillos, pero cuando el problema se complica, su falta de exploración se nota y la calidad de las soluciones disminuye de manera importante.
- El Greedy Aleatorio funciona de manera muy parecida, aunque introduce algo de variabilidad gracias al componente aleatorio. Esto le permite generar soluciones iniciales distintas y, en ocasiones, algo mejores que las del Greedy clásico. Aun así, la mejora no es constante ni demasiado grande, lo que demuestra que la aleatoriedad por sí sola no basta para resolver bien las instancias más difíciles.
- La Búsqueda Local tarda algo más que los dos métodos anteriores, pero a cambio ofrece una mejora notable en los resultados. A partir de las soluciones generadas por el Greedy Aleatorio, explora el vecindario mediante intercambios y va encontrando configuraciones más eficientes. En general, consigue resultados cercanos al óptimo y se comporta de forma muy estable entre ejecuciones, por lo que representa un buen equilibrio entre tiempo y calidad.
- Por último, la Búsqueda Tabú es la más lenta, pero también la más potente. Gracias a las memorias, los mecanismos de aspiración y las estrategias de intensificación y diversificación, logra escapar de los óptimos locales y explorar regiones que otros algoritmos no alcanzan. De esta manera obtiene soluciones muy cercanas o incluso iguales al óptimo, aunque requiere un mayor tiempo de cálculo.


En resumen, los cuatro algoritmos siguen una progresión natural: el Greedy y el Greedy Aleatorio destacan por su rapidez, la Búsqueda Local logra un equilibrio muy razonable entre velocidad y calidad, y la Búsqueda Tabú, aunque más lenta, es la que consigue los mejores resultados. Esto confirma la idea general de las metaheurísticas: cuanto más compleja es la estrategia de exploración, más tiempo necesita el algoritmo, pero también más cerca está de la mejor solución posible.

Finalmente vamos a mostrar una gráfica donde se observa claramente el coste medio por cada algoritmo e instancia de datos:



## Logs enlace drive:

Los logs están en el siguiente enlace:

 [logs](#)