Metaheurísticas (Curso 2021-2022)

Grado en Ingeniería Informática
Universidad de Granada

UNIVERSIDAD DE GRANADA

Práctica 3: Búsquedas por Trayectorias

Problema a: Mínima Dispersión Diferencial

Pedro Bedmar López - 75935296Z
pedrobedmar@correo.ugr.es

Grupo de prácticas 3 - Martes 17:30-19:30

Índice

Parte I

Formulación del problema

Sea $G = (V, E)$ un grafo completo no dirigido donde V, de tamaño n, es el conjunto de vértices que lo forman y E es el conjunto de las aristas que unen estos vértices. Este grafo es un grafo ponderado, ya que cada una de las aristas $e_{u,v} \in E$ lleva asociada un peso que representa la distancia $d_{u,v}$ entre dos vértices $u, v \in V$.

La dispersión es una medida que se puede aplicar en este dominio, donde dado un subconjunto $S \subset V$ de tamaño m se mide cómo de homogéneas son las distancias entre los vértices que forman S. Una de las aplicaciones más importantes de las Ciencias de la Computación consiste en optimizar valores como éste, maximizando o minimizando el resultado que devuelve una **función objetivo**.

En esta práctica queremos minimizar su valor, obteniendo la mínima dispersión. Este problema tiene un gran paralelismo con problemas reales, como puede ser la organización del género en almacenes, donde minimizar la dispersión de la mercancía reduce los costes. Por tanto, si resolvemos este problema de forma teórica es trivial aplicar la solución en estos casos.

Anteriormente he definido la dispersión de una forma muy genérica, sin entrar en su formalización. Y es que se puede definir de diferentes formas, teniendo en cuenta la dispersión media de los elementos del conjunto S o utilizando los valores extremos (máximos y mínimos) en éste. Esta segunda opción se define formalmente como:

$$diff(S) = max_{i \in S}\{\sum_{j \in S} d_{i,j}\} - min_{i \in S}\{\sum_{j \in S} d_{i,j}\}$$

Utilizando esta definición de dispersión como función objetivo obtenemos lo que se conoce como **Problema de la Mínima Dispersión Diferencial (MDD)**, es decir:

$$S^* = argmin_{S \subset V} diff(S)$$

Parte II

Descripción de la aplicación de los algoritmos

En la tercera práctica de la asignatura implementamos los algoritmos de **Enfriamiento Simulado (ES)**, **Búsqueda multiarranque básica (BMB)**, **Búsqueda Local Reiterada (ILS)** e **Hibridación de ILS y ES (ILS-ES)**, y comparamos su rendimiento entre ellos y con los algoritmos Greedy y Búsqueda Local de la primera práctica. Antes de describirlos, vamos a comentar información común a ambos.

1. Datos utilizados

Los datos que necesitamos para analizar el comportamiento de los algoritmos en este problema no son muy complejos. En cada posible instancia se necesita conocer el valor n indicando el número de puntos que contiene el dataset, el valor $m < n$ indicando cuantos puntos se quieren escoger de forma que se minimice la dispersión en esos m puntos y la matriz d con tamaño $n \times n$, simétrica y con valor 0 en su diagonal, que contiene las distancias entre cada uno de los n puntos del dataset. En definitiva, se necesita conocer el grafo G.

En total, en los experimentos utilizamos 50 instancias diferentes con datos extraídos del dataset **GKD**. Las instancias toman valores $n \in \{25, 50, 100, 125, 150\}$ y $m \in [2, 45]$.

2. Representación de soluciones

El conjunto V descrito en la formulación del problema coincide con n en tamaño. S es una solución válida del problema si:

- $|S| = m$
- $S \subset V$

Y por tanto, $m < n$.

3. Función objetivo

Como hemos comentado al describir el problema, la función objetivo a minimizar se define como:

$$diff(S) = max_{i \in S}\{\sum_{j \in S} d_{i,j}\} - min_{i \in S}\{\sum_{j \in S} d_{i,j}\}$$

En pseudocódigo quedaría de la siguiente forma:

Algorithm 1 Función objetivo

```
max ← −∞
min ← ∞
for s ∈ S do
    distance ← Σ_{s2∈S} d_{s,s2}
    if distance > max then
        max ← distance
    end if
    if distance < min then
        min ← distance
    end if
end for
return max − min
```

Tanto en la Búsqueda Local como en el Enfriamiento Simulado, no se utiliza directamente esta implementación de la funcion objetivo (excepto para inicializarlos). Esto se debe a que es costosa, en concreto tiene una complejidad computacional de $O(n^2)$. Utilizamos versiones factorizadas de la función, que reutilizan cálculos previos de iteraciones anteriores para actualizar el valor de la dispersión. De esta forma, obtenemos una complejidad de $O(n)$.

4. Generador de soluciones aleatorias

Utilizado en diferentes algoritmos para generar soluciones iniciales.

Algorithm 2 Generador de soluciones aleatorias en representación entera.

```
1:  procedure generateRandomSolution
2:      U ← [ ]
3:      S ← [ ]
4:
5:      for i = 0 to n − 1 do
6:          U ← U + {i}
7:      end for
8:
9:      shuffle(U)
10:     for i = 0 to m − 1 do
11:         aux ← U.last
12:         U ← U − {U.last}
13:         S ← S + {aux}
14:     end for
15:
16:     return U, S
17: end procedure
```

Parte III

Pseudocódigo de los algoritmos

A diferencia de los algoritmos genéticos, en los algoritmos basados en trayectorias se parte de una única solución y se inicia una búsqueda que sigue una trayectoria en el espacio de soluciones. En algunos de estos algoritmos, se realizan diferentes ejecuciones desde distintos puntos de inicio con el objetivo de evitar óptimos locales. A continuación, mostramos el pseudocódigo de los cuatro algoritmos que se implementan en la práctica junto con el de la búsqueda local.

5. Búsqueda Local (BL)

Algorithm 3 Algoritmo Búsqueda Local primero el mejor. Dada una solución inicial, se explora su entorno devolviendo la mejor solución encontrada en un número de evaluaciones determinado. El cálculo de la dispersión se realiza de forma factorizada.

```
1:  procedure localSearch(U, S, current_cost, max_eval)
2:      best_S ← S
3:      best_U ← U
4:      best_cost ← current_cost
5:
6:      sum ← [ ]                                   ▷ Array inicializado a 0
7:      for w ∈ S do
8:          for w2 ∈ S do
9:              sum[w] ← d_{w,w2}
10:         end for
11:     end for
12:
13:     eval ← 0
14:     better_solution ← true
15:
16:     while eval < max_eval and better_solution do
17:         better_solution ← false
18:
19:         for u ∈ S and while !better_solution and eval < max_eval do
20:             for v ∈ U and while !better_solution and eval < max_eval do
21:                 eval ← eval + 1
22:
```

```
23:
24:             δ ← [ ]                                   ▷ Array inicializado a 0
25:             δ(w)_max ← −∞
26:             δ(w)_min ← ∞
27:
28:             for w ∈ S do
29:                 if w! = u then
30:                     δ[w] ← sum[w] − d_{w,u} + d_{w,v}
31:                     δ[v] += d_{w,v}
32:
33:                     if δ[w] > δ(w)_max then
34:                         δ(w)_max ← δ[w]
35:                     end if
36:                     if δ[w] < δ(w)_min then
37:                         δ(w)_min ← δ[w]
38:                     end if
39:                 end if
40:             end for
41:
42:             δ_max ← max(δ[v], δ(w)_max)
43:             δ_min ← min(δ[v], δ(w)_min)
44:             new_cost ← δ_max − δ_min
45:             if new_cost < current_cost then
46:                 best_cost ← new_cost
47:                 current_cost ← new_cost
48:
49:                 swap ← u                              ▷ intercambio u y v en S y U
50:                 u ← v
51:                 v ← swap
52:                 better_solution = true
53:                 best_S = S
54:                 best_U = U
55:             end if
56:
57:         end for
58:     end for
59:
60:     shuffle(S)
61:     shuffle(U)
62: end while
63:
64: return U, S, best_cost
65: end procedure
```

6. Enfriamiento Simulado (ES)

Algorithm 4 Algoritmo de enfriamiento simulado. Parte de una solución inicial e itera buscando soluciones vecinas que mejoren a la actual. Además, tiene la capacidad de aceptar soluciones que no cumplen esta condición de forma aleatoria, pero conforme avanza el algoritmo, la probabilidad de aceptar aleatoriamente se disminuye.

```
1:  procedure SIMULATEDANNEALING(U, S, current_cost, max_eval)
2:      best_S ← S
3:      best_U ← U
4:      best_cost ← current_cost
5:
6:      max_nb ← 5 * n
7:      max_successes ← 0,1 * max_nb
8:      M ← max_eval/max_nb
9:      eval ← 0
10:
11:     sum ← [ ]                                   ▷ Array inicializado a 0
12:     for w ∈ S do
13:         for w2 ∈ S do
14:             sum[w] ← d_{w,w2}
15:         end for
16:     end for
17:
18:     t0 ← (0,3 * current_cost)/(−log(0,3))       ▷ Cálculo de la temperatura inicial
19:     tf ← 0,001
20:     while t0 <= tf and current_cost! = 0 do     ▷ Su valor debe ser menor que tf
21:         tf ← tf/10
22:     end while
23:
24:     beta ← (t0 − tf)/(0,3 * t0 * tf)
25:     t ← t0
26:     while eval < max_eval do
27:         count_nb ← 0
28:         count_successes ← 0
29:
30:         while count_nb < max_nb and count_successes < max_successes do
31:             index_s ← randint(0, m − 1)
32:             index_u ← randint(0, n − m − 1)
33:             swap_s ← S[index_s]
34:             swap_u ← U[index_u]
35:
36:             δ ← [ ]                             ▷ Array inicializado a 0
37:             δ(w)_max ← −∞
38:             δ(w)_min ← ∞
```

```
39:
40:             for w ∈ S do
41:                 if w! = swap_s then
42:                     δ[w] ← sum[w] − d_{w,swap_s} + d_{w,swap_u}
43:                     δ[swap_u] += d_{w,swap_u}
44:
45:                     if δ[w] > δ(w)_max then
46:                         δ(w)_max ← δ[w]
47:                     end if
48:                     if δ[w] < δ(w)_min then
49:                         δ(w)_min ← δ[w]
50:                     end if
51:                 end if
52:             end for
53:
54:             δ_max ← max(δ[swap_u], δ(w)_max)
55:             δ_min ← min(δ[swap_u], δ(w)_min)
56:             new_cost ← δ_max − δ_min
57:
58:             δ_f = fabs(neighbout_cost − current_cost)
59:             count_nb ← count_nb + 1
60:             eval ← eval + 1
61:
62:             if neighbout_cost < current_cost or randFloat(0, 1) <= e^{(−δ_f/t)} then
63:                 S[index_s] ← swap_u
64:                 U[index_u] = swap_s
65:                 sum ← delta
66:                 current_cost ← neighbout_cost
67:                 count_successes ← count_successes + 1
68:
69:                 if current_cost < best_cost then
70:                     best_S ← S
71:                     best_U ← U
72:                     best_cost ← current_cost
73:                 end if
74:
75:             end if
76:         end while
77:         t ← t/(1 + beta ∗ t)                    ▷ Enfriamiento de la temperatura
78:
79:     end while
80:
81:     return U, S, best_cost
82: end procedure
```

7. Búsqueda multiarranque básica (BMB)

Algorithm 5 Algoritmo de la búsqueda multiarranque básica. En vez de ejecutar la búsqueda local una única vez hasta las 100000 evaluaciones, se ejecuta 10 veces hasta las 10000 evaluaciones cada vez. Finalmente se devuelve el resultado de la mejor ejecución. En cada una se parte de una solución generada aleatoriamente.

```
1:  procedure BMB
2:      T ← 10
3:      maxEvalsLS ← 100000/T
4:
5:      best_cost ← ∞
6:      for i = 0 to T − 1 do
7:          U, S ← generateRandomSolution()
8:
9:          current_cost ← dispersion(S)
10:         U, S, current_cost ← localSearch(U, S, current_cost, maxEvalsLS)
11:
12:         if current_cost < best_cost then
13:             best_S ← S
14:             best_cost ← current_cost
15:         end if
16:     end for
17:
18:     return best_S
19: end procedure
```

8. Búsqueda Local Reiterada (ILS)

Algorithm 6 Algoritmo de la búsqueda local reiterada. Se parte de una solución generada aleatoriamente a la que se aplica la búsqueda local y se almacena la solución. Se inicia un proceso iterativo en el que se aplica una mutación a la solución anterior, se le vuelve a aplicar la búsqueda local, y si la solución mejora a la anterior, la sustituye. Finalmente, se devuelve la almacenada en la última iteración.

```
1:  procedure ILS
2:      T ← 10
3:      maxEvalsLS ← 100000/T
4:      num_mutations ← 0,1 * m
5:      if num_mutations < 1 then
6:          num_mutations ← 1
7:      end if
8:
9:      U, S ← generateRandomSolution()
10:
11:     best_cost ← dispersion(S)
12:     localSearch(U, S, best_cost, maxEvalsLS)
13:     T ← T − 1
14:
15:     for i = 0 to T − 1 do
16:         newS ← shuffle(S)
17:         newU ← shuffle(U)
18:
19:         for j = 0 to num_mutations − 1 do          ▷ Operador de mutación
20:             swap ← newS[j]
21:             newS ← newU[j]
22:             newU ← swap
23:         end for
24:
25:         new_cost ← dispersion(newS)
26:         newU, newS, new_cost ← localSearch(newU, newS, new_cost, maxEvalsLS)
27:
28:         if new_cost < best_cost then
29:             bestS ← newS
30:             bestU ← newU
31:             best_cost ← new_cost
32:         end if
33:     end for
34:
35:     return bestS
36: end procedure
```

9. Hibridación de ILS y ES (ILS-ES)

Algorithm 7 Algoritmo ILS hibridado con enfriamiento simulado. Realiza el mismo proceso que el algoritmo ILS, utilizando el enfriamiento simulado en vez de la búsqueda local.

```
1: procedure ILS-ES
2:     T ← 10
3:     maxEvalsES ← 100000/T
4:     num_mutations ← 0,1 * m
5:     if num_mutations < 1 then
6:         num_mutations ← 1
7:     end if
8:
9:     U, S ← generateRandomSolution()
10:
11:    best_cost ← dispersion(S)
12:    simulatedAnnealing(U, S, best_cost, maxEvalsES)
13:    T ← T − 1
14:
15:    for i = 0 to T − 1 do
16:        newS ← shuffle(S)
17:        newU ← shuffle(U)
18:
19:        for j = 0 to num_mutations − 1 do
20:            swap ← newS[j]
21:            newS ← newU[j]
22:            newU ← swap
23:        end for
24:
25:        new_cost ← dispersion(newS)
26:        newU, newS, new_cost ← simulatedAnnealing(newU, newS, new_cost, maxEvalsES)
27:
28:        if new_cost < best_cost then
29:            bestS ← newS
30:            bestU ← newU
31:            best_cost ← new_cost
32:        end if
33:    end for
34:
35:    return bestS
36: end procedure
```

Parte IV

Procedimiento considerado para desarrollar la práctica

La implementación de los algoritmos ha sido realizada en C++, concretamente en su versión de 2017. Para ello, hemos creado un proyecto con la siguiente estructura:

```
/
├── BIN ..... archivos ejecutables
│   ├── practica1
│   ├── practica2
│   └── practica3
├── data ..... ficheros .txt con los datos de entrada
│   ├── data_index.txt ..... índice con los nombres de los archivos de datos
│   └── ...
├── doc
├── FUENTES
│   ├── DataLoader.cpp ..... clase encargada de cargar los datos de los ficheros
│   ├── DataLoader.h
│   ├── functions.cpp ..... funciones auxiliares
│   ├── functions.h
│   ├── GreedyAlgorithm.cpp ..... implementación del algoritmo Greedy
│   ├── GreedyAlgorithm.h
│   ├── LocalSearchAlgorithm.cpp ..... implementación del algoritmo BL
│   ├── LocalSearchAlgorithm.h
│   ├── GeneticAlgorithm.cpp ..... implementación del algoritmo Genético
│   ├── GeneticAlgorithm.h
│   ├── MemeticAlgorithm.cpp ..... implementación del algoritmo Memético
│   ├── MemeticAlgorithm.h
│   ├── TrajectoryAlgorithms.cpp ..... impl. de los alg. basados en trayectorias
│   ├── TrajectoryAlgorithms.h
│   ├── practica1.cpp
│   ├── practica2.cpp
│   └── practica3.cpp ..... archivo desde donde se inicia la ejecución
├── obj ..... ficheros objeto
├── makefile
└── LEEME
```

Se ha partido desde cero, sin utilizar ningún framework de metaheurísticas ni librería adicional a las que vienen incluídas en el propio C++. Para la generación de números aleatorios, se utiliza la librería $< random >$ incluida en el lenguaje. La semilla utilizada en los experimentos es el número 1. El equipo donde se han realizado las

pruebas es un MacBook Pro de 15 pulgadas del año 2015, con CPU Intel Core i7 2.5 GHz I7-4870HQ y 16 GB de RAM. Utiliza el sistema operativo macOS Big Sur 11.6.1.

Para ejecutar el código, nos situamos en la raíz del proyecto y ejecutamos *make clean* y a continuación *make* en la terminal. Finalmente, ejecutamos:

$./bin/practica3 <semilla> <algoritmo> <fichero\_datos>$

Donde $<algoritmo>$ puede tomar como valor ES (Enfriamiento Simulado), BMB (Búsqueda Multiarranque Básica), ILS (Búsqueda Local Reiterada) o ILS-ES (Hibridación de ILS y ES).

Ejemplo: ./bin/practica3 1 ES data/GKD-b_50_n150_m45.txt

Parte V

Experimentos y análisis de resultados

Para comprobar el funcionamiento de los algoritmos, realizamos experimentos de ejecución. Nuestros algoritmos no son determinísticos, ya que la aleatoriedad está presente en ellos. Por tanto, para que los resultados sean reproducibles es necesario fijar una semilla. Como mencionamos al principio de la práctica, fijamos su valor en 1.

Vamos a ejecutar cada algoritmo con los 50 casos que tenemos. En la tabla siguiente se muestran los resultados medios de ejecución de cada algoritmo en cada uno de esos problemas.

Tabla 9.1: Resultados medios de cada algoritmo

Algoritmo	**Desviación**	**Tiempo**
Greedy	66.71	0.00006
BL	55.62	0.026
ES	54.86	0.001
BMB	41.5	0.02
ILS	50.08	0.01
ILS-ES	41.89	0.01

Como cabía esperar, el peor resultado viene dado por el algoritmo greedy. Para este problema, el criterio heurístico que utiliza no es demasiado bueno. La búsqueda local se comporta bien tanto en tiempo de ejecución como en desviación de la solución óptima, pero es mejorable ya que por su naturaleza puede atascarse en óptimos locales.

Por ello surgen algoritmos basados en trayectorias como el enfriamiento simulado, donde se permiten movimientos hacia soluciones peores. Mientras que la búsqueda local revisa el vecindario y sólo acepta soluciones que mejoran la encontrada hasta ahora, el enfriamiento simulado acepta de forma aleatoria soluciones peores.

El nombre de enfriamiento simulado proviene de una analogía con la termodinámica. En el algoritmo, existe una variable temperatura que va disminuyendo su valor conforme se desarrolla la ejecución. A menor temperatura, menor probabilidad de aceptar una solución peor. Por ello, este algoritmo equilibra la exploración al principio de su ejecución con la explotación al final.

En la práctica, para este problema no encontramos diferencias muy marcadas entre BL y ES, siendo ES un poco mejor que BL.

BMB es un algoritmo que realiza varias ejecuciones de la búsqueda local. Para

realizar una comparativa justa con BL, el número total de evaluaciones de la función objetivo siempre es el mismo, 100000, pero en BMB estas evaluaciones se dividen entre las 10 ejecuciones que realiza internamente de la búsqueda local. O sea, cada ejecución interna de la búsqueda local realizará 10000 evaluaciones. Aunque cada una de ellas profundice en menor medida en el espacio de búsqueda, tiene la ventaja de que cada ejecución se inicia desde diferentes puntos de este espacio, lo que consigue evitar óptimos locales o soluciones iniciales muy alejadas del óptimo. En una única ejecución de BL con 100000 evaluaciones nos arriesgamos a quedar atrapados en un óptimo local, con BMB esta búsqueda se diversifica desde diferentes puntos de inicio, lo que nos permite quedarnos con la mejor solución obteniendo únicamente 41.5 puntos de desviación.

ILS sigue una idea parecida a BMB, pero en este caso la solución inicial de cada ejecución de búsqueda local se basa en la solución de la ejecución anterior. A esta se le aplica una mutación, en nuestro caso únicamente al 10 % de los genes. Se puede observar que los resultados son peores que con BMB, se obtienen 50.08 puntos de desviación. Esto se debe a que aunque se realiza una mutación, la solución obtenida al realizar una ejecución de la búsqueda local sigue dependiendo en gran medida de la anterior. Elevar el porcentaje de genes mutados podría ser una solución para introducir más diferencias.

Finalmente, ILS-ES es una hibridación de ILS con enfriamiento simulado. Devuelve una buena desviación (41.89), similar a la que se obtiene con BMB. En comparación con ILS se obtienen mejores resultados, esto puede deberse a que de por sí ES ya incorpora una fase de exploración que compensa la falta que tenía BL en ILS.

Por tanto, BMB e ILS-ES son los algoritmos que devuelven un mejor resultado para este problema, por su equilibrio entre exploración y explotación.

En cuanto a tiempos de ejecución, todos resuelven la tarea en un tiempo similar. Este tiempo es bajo, ya que tanto BL como ES realizan un cálculo factorizado de la función objetivo. Este tiempo es aún menor en Greedy, ya que realiza una construcción incremental de la solución y termina cuando esta ha alcanzado los m elementos.

10. Resultados completos

Búsqueda Local

Tabla 10.1: Ejecución de la BL en cada problema

Caso	Coste medio obtenido	Desv	Tiempo
GKD-b_1_n25_m2	0.0000	0.00	1.58E-02
GKD-b_2_n25_m2	0.0000	0.00	1.62E-02
GKD-b_3_n25_m2	0.0000	0.00	1.58E-02
GKD-b_4_n25_m2	0.0000	0.00	1.51E-02
GKD-b_5_n25_m2	0.0000	0.00	1.56E-02
GKD-b_6_n25_m7	26.4899	51.99	1.65E-02
GKD-b_7_n25_m7	29.5395	52.27	1.33E-02
GKD-b_8_n25_m7	34.3999	51.28	1.36E-02
GKD-b_9_n25_m7	39.7469	57.06	1.35E-02
GKD-b_10_n25_m7	35.6030	34.65	1.63E-02
GKD-b_11_n50_m5	16.0448	88.00	2.22E-02
GKD-b_12_n50_m5	14.0267	84.88	2.48E-02
GKD-b_13_n50_m5	16.1494	85.37	2.18E-02
GKD-b_14_n50_m5	11.0517	84.95	3.00E-02
GKD-b_15_n50_m5	16.2757	82.47	2.87E-02
GKD-b_16_n50_m15	121.9782	64.96	4.24E-02
GKD-b_17_n50_m15	109.6692	56.13	3.53E-02
GKD-b_18_n50_m15	68.8174	37.23	3.47E-02
GKD-b_19_n50_m15	143.7370	67.71	3.22E-02
GKD-b_20_n50_m15	96.4486	50.53	3.48E-02
GKD-b_21_n100_m10	46.0147	69.94	3.90E-02
GKD-b_22_n100_m10	42.0094	67.47	3.19E-02
GKD-b_23_n100_m10	36.0227	57.40	2.76E-02
GKD-b_24_n100_m10	37.7987	77.14	2.82E-02
GKD-b_25_n100_m10	44.9021	61.69	2.65E-02
GKD-b_26_n100_m30	387.6356	56.47	3.48E-02
GKD-b_27_n100_m30	376.6248	66.25	3.15E-02
GKD-b_28_n100_m30	379.9123	72.00	3.18E-02
GKD-b_29_n100_m30	344.0501	60.05	3.03E-02
GKD-b_30_n100_m30	304.0676	58.08	3.01E-02
GKD-b_31_n125_m12	48.7661	75.92	2.24E-02
GKD-b_32_n125_m12	49.8411	62.30	2.29E-02
GKD-b_33_n125_m12	68.7881	73.06	2.38E-02
GKD-b_34_n125_m12	51.6503	62.27	2.39E-02
GKD-b_35_n125_m12	61.8754	70.73	2.34E-02
GKD-b_36_n125_m37	403.2151	61.45	3.25E-02
GKD-b_37_n125_m37	418.0820	52.43	3.41E-02
GKD-b_38_n125_m37	446.9816	57.95	3.34E-02
GKD-b_39_n125_m37	411.6134	59.04	3.27E-02
GKD-b_40_n125_m37	386.2099	53.86	3.34E-02
GKD-b_41_n150_m15	69.7972	66.55	1.99E-02
GKD-b_42_n150_m15	65.2854	58.97	2.11E-02
GKD-b_43_n150_m15	65.6087	59.22	2.06E-02
GKD-b_44_n150_m15	61.1381	57.58	2.05E-02
GKD-b_45_n150_m15	56.5077	50.85	1.99E-02
GKD-b_46_n150_m45	575.4527	60.42	3.52E-02
GKD-b_47_n150_m45	399.5420	42.78	3.50E-02
GKD-b_48_n150_m45	520.6545	56.45	4.16E-02
GKD-b_49_n150_m45	457.3357	50.49	3.58E-02
GKD-b_50_n150_m45	528.6708	52.93	3.49E-02

ES y BMB

Tabla 10.2: Ejecución de ES y BMB en cada problema

Caso	ES			BMB		
	Coste obtenido	**Desv**	**Tiempo**	**Coste obtenido**	**Desv**	**Tiempo**
GKD-b_1_n25_m2	0.0000	0.00	0.0001	0.00	0.00	0.00
GKD-b_2_n25_m2	0.0000	0.00	0.0001	0.00	0.00	0.00
GKD-b_3_n25_m2	0.0000	0.00	0.0001	0.00	0.00	0.00
GKD-b_4_n25_m2	0.0000	0.00	0.0001	0.00	0.00	0.00
GKD-b_5_n25_m2	0.0000	0.00	0.0001	0.00	0.00	0.00
GKD-b_6_n25_m7	30.8142	58.73	0.0001	15.29	16.80	0.00
GKD-b_7_n25_m7	31.3085	54.97	0.0001	20.02	29.59	0.00
GKD-b_8_n25_m7	47.5303	64.74	0.0001	20.96	20.03	0.00
GKD-b_9_n25_m7	17.0692	0.00	0.0001	25.54	33.17	0.00
GKD-b_10_n25_m7	36.4859	36.23	0.0002	26.88	13.46	0.00
GKD-b_11_n50_m5	3.9312	51.00	0.0002	11.55	83.32	0.00
GKD-b_12_n50_m5	9.6897	78.11	0.0002	7.05	69.93	0.00
GKD-b_13_n50_m5	15.9943	85.23	0.0002	13.27	82.20	0.00
GKD-b_14_n50_m5	22.6287	92.65	0.0002	4.87	65.85	0.00
GKD-b_15_n50_m5	23.6894	87.96	0.0002	11.46	75.11	0.00
GKD-b_16_n50_m15	144.7402	70.47	0.0005	73.59	41.92	0.00
GKD-b_17_n50_m15	116.4918	58.70	0.0003	83.23	42.20	0.01
GKD-b_18_n50_m15	67.6273	36.13	0.0003	57.32	24.65	0.01
GKD-b_19_n50_m15	133.9178	65.34	0.0003	83.22	44.23	0.00
GKD-b_20_n50_m15	93.4749	48.95	0.0003	76.31	37.48	0.00
GKD-b_21_n100_m10	43.4814	68.19	0.0006	17.50	20.94	0.01
GKD-b_22_n100_m10	86.8405	84.27	0.0004	30.86	55.72	0.01
GKD-b_23_n100_m10	37.7569	59.36	0.0004	25.78	40.48	0.01
GKD-b_24_n100_m10	58.9135	85.33	0.0005	27.59	68.68	0.01
GKD-b_25_n100_m10	44.9492	61.73	0.0005	25.38	32.24	0.01
GKD-b_26_n100_m30	335.5107	49.71	0.0012	285.68	40.94	0.03
GKD-b_27_n100_m30	314.4841	59.59	0.0014	215.02	40.89	0.03
GKD-b_28_n100_m30	246.5015	56.84	0.0018	277.20	61.62	0.03
GKD-b_29_n100_m30	416.4145	66.99	0.0008	248.88	44.77	0.03
GKD-b_30_n100_m30	317.1722	59.81	0.0011	262.70	51.47	0.03
GKD-b_31_n125_m12	57.3335	79.51	0.0007	27.70	57.60	0.02
GKD-b_32_n125_m12	43.7520	57.06	0.0006	35.15	46.55	0.01
GKD-b_33_n125_m12	50.6182	63.39	0.0007	27.48	32.56	0.01
GKD-b_34_n125_m12	47.6088	59.07	0.0006	34.10	42.85	0.01
GKD-b_35_n125_m12	58.0287	68.79	0.0004	29.47	38.53	0.01
GKD-b_36_n125_m37	466.2217	66.66	0.0007	326.64	52.41	0.04
GKD-b_37_n125_m37	464.5845	57.19	0.0017	332.59	40.20	0.04
GKD-b_38_n125_m37	463.5547	59.45	0.0013	419.64	55.21	0.04
GKD-b_39_n125_m37	333.4804	49.45	0.0010	305.47	44.81	0.03
GKD-b_40_n125_m37	455.9530	60.92	0.0015	353.24	49.55	0.03
GKD-b_41_n150_m15	47.2989	50.64	0.0008	47.43	50.77	0.01
GKD-b_42_n150_m15	94.9071	71.77	0.0008	46.20	42.02	0.01
GKD-b_43_n150_m15	137.1540	80.49	0.0005	46.17	42.06	0.02
GKD-b_44_n150_m15	67.8017	61.75	0.0005	40.31	35.66	0.02
GKD-b_45_n150_m15	38.9101	28.62	0.0012	49.26	43.62	0.02
GKD-b_46_n150_m45	563.2659	59.57	0.0020	483.23	52.87	0.03
GKD-b_47_n150_m45	403.5339	43.35	0.0027	388.95	41.23	0.05
GKD-b_48_n150_m45	501.1548	54.76	0.0034	494.48	54.14	0.04
GKD-b_49_n150_m45	652.6593	65.31	0.0014	558.58	59.47	0.05
GKD-b_50_n150_m45	700.7514	64.49	0.0029	558.43	55.44	0.04

ILS e ILS-ES

Tabla 10.3: Ejecución de la ILS e ILS-ES en cada problema

Caso	ILS			ILS-ES		
	Coste obtenido	**Desv**	**Tiempo**	**Coste obtenido**	**Desv**	**Tiempo**
GKD-b_1_n25_m2	0.0000	0.00	0.0001	0.0000	0.00	0.0003
GKD-b_2_n25_m2	0.0000	0.00	0.0001	0.0000	0.00	0.0002
GKD-b_3_n25_m2	0.0000	0.00	0.0002	0.0000	0.00	0.0002
GKD-b_4_n25_m2	0.0000	0.00	0.0001	0.0000	0.00	0.0002
GKD-b_5_n25_m2	0.0000	0.00	0.0001	0.0000	0.00	0.0004
GKD-b_6_n25_m7	27.3520	53.50	0.0003	25.8947	50.89	0.0013
GKD-b_7_n25_m7	25.6926	45.13	0.0003	19.6142	28.12	0.0014
GKD-b_8_n25_m7	39.5138	57.58	0.0002	20.9589	20.03	0.0010
GKD-b_9_n25_m7	38.9251	56.15	0.0003	17.0692	0.00	0.0013
GKD-b_10_n25_m7	44.5938	47.83	0.0003	26.8843	13.46	0.0012
GKD-b_11_n50_m5	14.9851	87.15	0.0006	3.9312	51.00	0.0030
GKD-b_12_n50_m5	14.7687	85.64	0.0007	7.6237	72.18	0.0034
GKD-b_13_n50_m5	14.5079	83.72	0.0007	8.9731	73.67	0.0037
GKD-b_14_n50_m5	8.0826	79.42	0.0007	6.7611	75.40	0.0027
GKD-b_15_n50_m5	15.3743	81.44	0.0007	8.6983	67.20	0.0040
GKD-b_16_n50_m15	87.0772	50.91	0.0016	112.7921	62.10	0.0057
GKD-b_17_n50_m15	131.4499	63.40	0.0015	83.6308	42.48	0.0067
GKD-b_18_n50_m15	67.7767	36.27	0.0017	67.6273	36.13	0.0063
GKD-b_19_n50_m15	170.4828	72.78	0.0018	66.5658	30.28	0.0065
GKD-b_20_n50_m15	100.1682	52.36	0.0017	83.7435	43.02	0.0061
GKD-b_21_n100_m10	48.1798	71.29	0.0033	34.0256	59.35	0.0086
GKD-b_22_n100_m10	34.5064	60.40	0.0026	36.3172	62.38	0.0070
GKD-b_23_n100_m10	57.5326	73.33	0.0026	21.1500	27.44	0.0087
GKD-b_24_n100_m10	46.1266	81.27	0.0027	34.6530	75.07	0.0066
GKD-b_25_n100_m10	54.8968	68.67	0.0025	29.1385	40.97	0.0067
GKD-b_26_n100_m30	179.5874	6.05	0.0226	241.6666	30.18	0.0135
GKD-b_27_n100_m30	293.1102	56.64	0.0242	264.2277	51.90	0.0134
GKD-b_28_n100_m30	237.0526	55.12	0.0271	226.3351	53.00	0.0117
GKD-b_29_n100_m30	187.0094	26.50	0.0371	303.5755	54.72	0.0118
GKD-b_30_n100_m30	214.2950	40.51	0.0406	207.0661	38.44	0.0122
GKD-b_31_n125_m12	65.6209	82.10	0.0068	22.7916	48.47	0.0117
GKD-b_32_n125_m12	45.3354	58.56	0.0050	37.3428	49.69	0.0110
GKD-b_33_n125_m12	50.3064	63.16	0.0051	40.7318	54.50	0.0109
GKD-b_34_n125_m12	50.2437	61.21	0.0059	43.1295	54.81	0.0135
GKD-b_35_n125_m12	83.1226	78.21	0.0055	30.6815	40.97	0.0112
GKD-b_36_n125_m37	317.9790	51.12	0.0432	342.1841	54.58	0.0148
GKD-b_37_n125_m37	299.1416	33.51	0.0537	381.1489	47.82	0.0201
GKD-b_38_n125_m37	262.5100	28.40	0.0545	315.6762	40.46	0.0173
GKD-b_39_n125_m37	273.0323	38.25	0.0548	313.1201	46.16	0.0181
GKD-b_40_n125_m37	301.8810	40.97	0.0514	300.3319	40.67	0.0172
GKD-b_41_n150_m15	51.7237	54.86	0.0078	47.2989	50.64	0.0142
GKD-b_42_n150_m15	46.2038	42.02	0.0091	50.8322	47.30	0.0121
GKD-b_43_n150_m15	71.9053	62.79	0.0072	32.1134	16.69	0.0080
GKD-b_44_n150_m15	71.0724	63.51	0.0077	60.9271	57.43	0.0117
GKD-b_45_n150_m15	53.4290	48.02	0.0078	41.7350	33.45	0.0116
GKD-b_46_n150_m45	476.8534	52.24	0.0488	524.1422	56.55	0.0243
GKD-b_47_n150_m45	378.1455	39.55	0.0410	396.7161	42.38	0.0277
GKD-b_48_n150_m45	382.8464	40.77	0.0366	422.8448	46.38	0.0347
GKD-b_49_n150_m45	410.1888	44.80	0.0417	488.6990	53.67	0.0278
GKD-b_50_n150_m45	340.9032	27.00	0.0414	524.6836	52.57	0.0233