PRÁCTICA 3
METAHEURÍSTICAS

PROBLEMA 1: MDD

UNIVERSIDAD DE GRANADA

David Muñoz Sánchez 07256819C

dmunozs14@correo.ugr.es **Grupo A3**

(Martes)

Algoritmos considerados: ES, BMB, ILS.

Índice

Índice **2**

Descripción del problema **3**

Descripción de los algoritmos empleados y consideraciones comunes a estos **3**

Función objetivo 4

Generación de soluciones aleatorias 6

Componentes particulares de cada algoritmo **7**

Esquema de búsqueda 7

Enfriamiento simulado 7

BMB 8

ILS 9

Enfriamiento simulado 10

Cálculo Temperatura inicial 10

Esquema de enfriamiento 10

Búsqueda Local 11

Método creación de lista de candidatos 11

Exploración de entorno 11

Generación de vecino 12

ILS 12

Operador de mutación 12

Procedimiento desarrollo **13**

Experimentos y análisis de resultados **14**

Descripción del problema

El problema es el MDD (*Minimum Differential Dispersion Problem)*, que consiste en seleccionar un conjunto de m elementos de un conjunto inicial de n elementos de forma que se minimice la dispersión entre los elementos escogidos, de los que tenemos las distancias entre ellos.

La dispersión es la diferencia entre la distancia acumulada máxima y la mínima de un conjunto de m elementos. La distancia acumulada de un elemento es la suma de distancias de ese elemento a todos los demás.

A partir de una solución inicial aleatoria, o, en el caso de esta práctica, varias soluciones aleatorias, se tiene que conseguir una buena solución.

Descripción de los algoritmos empleados y consideraciones comunes a estos

Antes de comenzar con la explicación de los algoritmos y las consideraciones comunes a estos es necesario indicar el esquema de representación de soluciones. Todas las versiones trabajan con conjuntos soluciones formados por números reales entre 0 y n (tamaño del problema). Las soluciones tienen todas tamaño m (m < n) y los elementos que forman la solución no se pueden repetir.

Se han implementado varias versiones de algoritmos que se basan en la búsqueda por trayectorias, que tienen como objetivo mejorar la búsqueda local, que se queda estancada en mínimos locales rápidamente.

El Enfriamiento Simulado parte de una solución aleatoria de la que generamos un vecino hasta máximo de vecinos. Cuando tenemos cierto número de éxitos, es decir, que aceptemos la solución o bien porque mejore la que tenemos ahora o por la función de probabilidad, también empezamos de nuevo con el proceso. Este proceso se repite hasta que no haya éxitos en una iteración o hasta que se llegue a las 100’000 evaluaciones de la función objetivo.

ILS y BMB sin embargo se basan en la búsqueda multiarranque. Partimos de una solución aleatoria y diez, respectivamente. Les vamos aplicando búsqueda local y siempre mantenemos la mejor solución encontrada hasta el momento. En ILS la búsqueda local se tiene que ejecutar 10 veces, las mismas que en BMB.

Función objetivo

Tenemos dos formas de calcular la función objetivo:

```
maximo ← 0
minimo ← 0
suma ← 0

para i ← 0 hasta TAMAÑO de elegidos[]
[incremento 1] hacer
      para j ← 0 hasta TAMAÑO de elegidos[]
      [incremento 1] hacer
            Si elegidos[i] <> elegidos[j] entonces
                  suma ← suma +
            distancias[elegidos[i]][elegidos[j]]
                  //La matriz de distancias se pasa como
            parámetro a la función objetivo
      fin_para
      Si suma > maximo entonces
            maximo ← suma
      Fin_si
      Si minimo = 0 entonces
            minimo ← suma
      Sino entonces
            Si suma < minimo entonces
                  minimo ← suma
            Fin_si
      Fin_si

      suma ← 0

fin_para

Devolver maximo - minimo
```

Esta primera forma es más costosa que la segunda, que se basa en la factorización y calcula el fitness a partir del vector de distancias acumuladas de una solución, la matriz de distancias y una solución vecina (es decir, que a nuestra solución se le haya aplicado un movimiento de intercambio en alguna posición).

```
maximo_nuevo ← 0
minimo_nuevo ← 0
suma ← 0
distancias_acumuladas[] //Float
```

```
     indice_cambio //Pasado como argumento, único índice en el que
difieren las dos soluciones

     para i ← 0 hasta TAMAÑO de distancias_acumuladas[]
     [incremento 1] hacer

          Si i <> indice_cambio entonces
               distancias_acumuladas[i] ←
     distancias_acumuladas[i]-matriz_distancias[solucion_actual[ind
     ice_cambio]][solucion_actual[i]]
          Fin_si

     fin_para


     para i ← 0 hasta TAMAÑO de distancias_acumuladas[]
     [incremento 1] hacer

          Si i <> indice_cambio entonces
               distancias_acumuladas[i] ←
     distancias_acumuladas[i]+matriz_distancias[solucion_vecina[ind
     ice_cambio]][solucion_actual[i]]
               suma ← suma +
     matriz_distancias[solucion_vecina[indice_cambio]][solucion_act
     ual[i]]
          Fin_si

     fin_para

     distancias_acumuladas[indice_cambio] ← suma

     para i ← 0 hasta TAMAÑO de distancias_acumuladas[]
     [incremento 1] hacer
          distancia ← distancias_acumuladas[i]

          Si distancia > maximo_nuevo entonces
               maximo_nuevo ← distancia
          Fin_si

          Si minimo_nuevo = 0 entonces
               minimo_nuevo ← distancia
          Sino entonces
               Si distancia < minimo_nuevo entonces
                    minimo_nuevo ← distancia
               Fin_si
          Fin_si
     fin_para
```

Devolver maximo_nuevo - minimo_nuevo

Generación de soluciones aleatorias

Tanto ES como ILS crean una única solución aleatoria que se hace de la siguiente forma:

primer_sel ← seleccionables[] ÚLTIMO ELEMENTO //Iremos sacando posibles números de nuestra solución de seleccionables, que es un vector con número de 0 a n-1
seleccionables[] ← quitar ÚLTIMO ELEMENTO
solucion_inicial[] ← añadir primer_sel
solucion_actual[] ← añadir primer_sel
mejor_solucion[] ← añadir primer_sel
//Se rellenan todos los vectores por comodidad
Mientras TAMAÑO DE solucion_inicial[] < m entonces
Random ← mezclar seleccionables[]
sel ← seleccionables[] ÚLTIMO ELEMENTO
seleccionables[] ← quitar ÚLTIMO ELEMENTO
solucion_inicial[] ← añadir sel
solucion_actual[] ← añadir sel
mejor_solucion[] ← añadir sel
Fin_mientras

El algoritmo con una forma un poco diferente de tratar las soluciones iniciales aleatorias es BMB, ya que debemos generar 10. Las 10 se construyen de forma que no se pueden repetir elementos entre ellas.

seleccionables_anterior[] ← seleccionables[]
Mientras contador_soluciones < NUM_SOLUCIONES entonces
Random ← mezclar seleccionables[]
primer_sel ← seleccionables[] ÚLTIMO ELEMENTO //Iremos sacando posibles números de nuestra solución de seleccionables, que es un vector con número de 0 a n-1
seleccionables[] ← quitar ÚLTIMO ELEMENTO
solucion_inicial[] ← añadir primer_sel
//Rellenamos solo solucion_inicial[] por la construcción del algoritmo
Mientras TAMAÑO DE solucion_inicial[] < m entonces
Random ← mezclar seleccionables[]
sel ← seleccionables[] ÚLTIMO ELEMENTO
seleccionables[] ← quitar ÚLTIMO ELEMENTO
solucion_inicial[] ← añadir sel
Fin_mientras

soluciones_iniciales[][] ← añadir solucion_inicial[]

```
            solucion_inicial[] ← limpiar
            contador_soluciones ← contador_soluciones + 1
            seleccionables[] ← seleccionables_anterior[]
      Fin_mientras
```

Componentes particulares de cada algoritmo

Esquema de búsqueda

A continuación, se muestran unos esquemas generales en forma de pseudocódigo para ilustrar cómo cada algoritmo intenta obtener la mejor solución posible. En dichos códigos, se van a obviar los procedimientos (se reflejaran como ES y BL), que se usan para buscar y se mostrará el tratamiento que se da a las soluciones que se van obteniendo, es decir, cómo se busca realmente.

Enfriamiento simulado

Partimos de una solución inicial aleatoria y se inicializan las variables necesarias para la correcta ejecución del algoritmo, tales como **max_vecinos**, **max_exitos**, **beta** (necesario para el enfriamiento), así como los vectores de distancias acumuladas.

```
      Hacer
            exitos ← 0
            distancias_originales ← distancias acumuladas
            //Guardo este vector porque en el fitness factorizado se
modifica y ya no tendríamos el vector de distancias de la solución
actual si no de la vecina

            para i ← 0 hasta max_vecinos and exitos < max_exitos
      [incremento 1] hacer
                  GENERACION VECINO Y CALCULO DE SU FITNESS
                  variacion_fitness ← fitness_nuevo - fitness_original
                  //En la primera iteración fitness_original vale
mejor_fitness que es el fitness de la solución inicial en ese
momento
                  valor_random ← Número random entre 0.0 y 1.0
                  valor prob ← exp((-1.0 * variacion_fitness) /
temperatura);

                  Si variacion_fitness < 0 or valor_random <=
valor_prob entonces
                        exitos ← exitos + 1
                        solucion_actual[] ← solucion_vecina[]
```

```
                    seleccionables[] ← quitamos ultimo elemento
                    //Lo hemos metido en la vecina
                    seleccionables[] ← añadir elemento quitado de
               la solucion_actual
                    fitness_original ← fitness_nuevo
                    distancias_originales[] ←
               distancias_acumuladas[]

                    Si fitness_nuevo < mejor_fitness entonces
                         mejor_solucion[] ← solucion_actual[]
                         mejor_fitness ← fitness_nuevo
                    Fin_si
               Si no entonces
                    distancias_acumuladas[] ←
distancias_originales[]
               Fin_si
          fin_para
```

LLAMADA AL PLANIFICADOR DE TEMPERATURA PARA ENFRIAR

Mientras cuenta_ev < EVALUACIONES and exitos <> 0

BMB

En este algoritmo partimos de un vector de soluciones aleatorio. A cada solución se le aplicará búsqueda local y mantendremos siempre la mejor solución obtenida.

para i ← 0 hasta NUM_SOLUCIONES [incremento 1] hacer

cuenta_ev ← 0

soluciones[] //Enteros
vecinos[] //Vector de pares

stuck ← true
solucion_actual[] ← soluciones_iniciales[i]
mejor_fitness ← ***Dispersión sin factorizar de solucion actual***
distancias_acumuladas[] ← ***Procedimiento para calcular las distancias***

BÚSQUEDA LOCAL() *La solución obtenida estará en solución actual*

Si primera entonces
mejor_solucion[] ← solucion_actual[]

```
            fitness_maximo ← mejor_fitness
        Fin_si
        Si mejor_fitness < fitness_maximo entonces
            mejor_solucion[] ← solucion_actual[]
            fitness_maximo ← mejor_fitness
        Fin_si
    fin_para
```

ILS

Este algoritmo también aplica 10 veces la búsqueda local, pero parte de una única solución inicial.

Primero se aplica búsqueda local a la inicial y obtenemos una solución, la cual vamos a mutar y aplicaremos búsqueda local a la mutación para después quedarnos con la solución original o lo que nos ha devuelto la BL sobre la mutada (según cual sea mejor).

```
    para i ← 0 hasta NUM_SOLUCIONES [incremento 1] hacer
        solucion_original[]
        fitness_original
        solucion_original[] ← solucion_actual[]
        solucion_actual_mutada[] ← MUTACIÓN SOBRE SOLUCION_ACTUAL
        fitness_original ← dispersión sin factorizar de
        solución_actual

        cuenta_ev ← 0

        soluciones[]
        soluciones_anterior[]
        vecinos[]

        stuck ← true

        Si first_ite entonces
            solucion_actual[] ← solucion_inicial[]
            first_ite ← false
        Sino entonces
            solucion_actual[] ← solucion_actual_mutada[]
        Fin_si

        mejor_fitness ← dispersión sin factorizar de
        solucion_actual
        distancias_acumuladas[] ← se calculan con procedimiento

        BÚSQUEDA LOCAL() //Nos interesa la solución que ha dejado

        Si primera entonces
            mejor_solucion[] ← solucion_actual[]
            solucion_maxima[] ← mejor_solucion[]
```

```
                fitness_maximo ← mejor_fitness
                primera ← false
          Sino entonces
                //Tenemos que elegir entre la actual (mutada a la
          que hemos aplicado BL) y la original
                Si mejor_fitness < fitness_original entonces
                      mejor_solucion[] ← solucion_actual[]
                Sino entonces
                      solucion_actual[] ← solucion_original[]
                      mejor_fitness ← fitness_original
                Fin_si

                //Actualizamos la mejor solución hasta ahora
                Si mejor_fitness < fitness_maximo entonces
                      solucion_maxima[] ← solucion_actual[]
                      fitness_maximo ← mejor_fitness
                Fin_si
          Fin_si
    fin_para
```

Enfriamiento simulado

Cálculo Temperatura inicial

La temperatura inicial se calcula según la fórmula que aparece en el guión de esta práctica.

```
t_inicial ← (0.3 * mejor_fitness) / (-1.0 * log(0.3))
//Mejor fitness se corresponde en esta sentencia con el
fitness de la solución inicial aleatoria que hemos generado
```

Esquema de enfriamiento

Del enfriamiento se encarga un método llamado t_scheduler, al que se le pasa el valor beta de la fórmula de enfriamiento y la temperatura que hay ahora mismo, por referencia.

Lo primero, beta se calcula de la siguiente forma:

```
beta ← (t_inicial - t_final) / ((EVALUACIONES / max_vecinos) *
t_final * t_inicial)
```

El método encargado de enfriar tiene la siguiente forma:

```
nueva_temperatura ← temperatura / (1 + beta * temperatura)
temperatura ← nueva_temperatura
```

Búsqueda Local

Método creación de lista de candidatos

```
vecinos[] ← limpiar //Vector de pares de enteros
soluciones[] <- limpiar

//El primer elemento de cada par es el elemento a sacar y el segundo el elemento a meter
para i ← 0 hasta TAMAÑO de solucion_actual[] [incremento 1] hacer
      soluciones[] ← añadir solucion_actual[i]
fin_para

sin_asignar[] //Vector de enteros

para i ← 0 hasta n [incremento 1] hacer
      Si i NO está en soluciones entonces
            sin_asignar[] ← añadir i
      Fin_si
      seleccionables[solucion_actual[i]] ← false
fin_para

para i ← 0 hasta TAMAÑO de soluciones[] [incremento 1] hacer
      para j ← 0 hasta TAMAÑO de sin_asignar[] [incremento 1] hacer
            vecinos[] ←añadir par(soluciones[i], sin_asignar[j])
      fin_para
fin_para
```

Exploración de entorno

```
distancias_originales[] ← distancias_acumuladas[]
nueva_dispersion
improved ← false
stuck ← true

para i ← 0 hasta TAMAÑO de vecinos[] and improved = false [incremento 1] hacer
      cambio ← vecinos[i]
      solucion_vecina ← genero vecino con cambio first y cambio second
      indice ← indice_cambio(solucion_actual, solucion_vecina)
      //Indice cambio se encarga de devolver la única posición en la que difieren la solución actual y su solución vecina
      nueva_dispersión ← fitness usando factorización
```

```
        cuenta_ev ← cuenta_ev + 1

        Si nueva_dispersion < mejor_fitness entonces
            improved ← true
            stuck ← false
            mejor_fitness ← nueva_dispersion
            solucion_actual[] ← solucion_vecina[]

        Sino entonces
            distancias_acumuladas[] ← distancias_originales[]
        Fin_si
    fin_para
```

Esta exploración del entorno se hace en un do_while por cada conjunto de vecinos que obtengamos dependiendo de cuál sea la solución actual en cada momento. El do_while termina cuando no obtenemos ningún vecino mejor en todo el entorno o cuando realizamos 100’000 evaluaciones de la función objetivo.

Generación de vecino

Se hace con un método al que se le pasa la solución actual, el elemento a sacar, el elemento a meter y el total de elementos.

```
    resultado[] ← solucion_actual[]
    pos_found ← -1

    para p ← 0 hasta TAMAÑO de resultado[] and pos_found < 0
    [incremento 1] hacer
        Si resultado[p] = elemento_a_sacar entonces
            pos_found ← p
        Fin_si
    fin_para

    resultado[pos_found] ← elemento_a_meter

    Devolver resultado[]
```

ILS

Operador de mutación

Hay que mutar exactamente 0.3 * m elementos de la solución.

```
    seleccionables[] ← tamaño n y todos los elementos a true
    indices_seleccionables[] ← tamaño m y todos los elementos a
true
```

```
indices_seleccionados[] //Enteros
resultado[] //Enteros

resultado[] ← solucion_actual
a_seleccionar ← 0.3 * m

para i ← 0 hasta TAMAÑO de solucion_actual[]
[incremento 1] hacer
      seleccionables[solucion_actual[i]] ← false
fin_para

para i ← 0 hasta a_seleccionar [incremento 1] hacer
      indice ← número random entre 0 y m-1

      Si indices_seleccioables[indice] = true entonces
            indices_seleccioados[] ← añadir indice
            indices_seleccionables[indice] ← false
      Fin_si
fin_para

para i ← 0 hasta TAMAÑO de indices_seleccionados[]
[incremento 1] hacer
      primer_elemento ← indices_seleccionados[i]
      elemento_a_meter ← número random entre 0 y n-1

      Mientras seleccionables[elemento_a_meter] = false
entonces
            elemento_a_meter ← número random entre 0 y n-1

      Fin_mientras

      traslado ← resultado[primer_elemento]
      resultado[primer_elemento] ← elemento_a_meter
      seleccionables[elemento_a_meter] ← false
      seleccionables[traslado] ← true
fin_para

Devolver resultado
```

Procedimiento desarrollo

La práctica ha sido desarrollada haciendo uso de C++. Para la implementación de la práctica, se ha tomado el código para leer los archivos con las distancias de la Práctica 1. Para leer los archivos, hago uso de las principales funciones de **fstream**, para el control de flujo y de errores. Se sigue la misma política que en

la práctica anterior de introducir la semilla para inicializar los números aleatorios como argumento de la ejecución (esta vez puede ser cualquier número, no solo entre 1 y 5).

Todo lo demás se ha hecho desde cero haciendo uso sobre todo del tipo de dato **vector** de la *STL* y, por consiguiente, de los métodos asociados a este así como de iteradores y sus métodos para borrar valores concretos de los vectores.

Experimentos y análisis de resultados

Es importante resaltar antes de proceder con el análisis, que todos los experimentos se han hecho introduciendo como argumento la **semilla 1.** No obstante, para las ejecuciones de Greedy y BL que se hicieron en la práctica 1, se hicieron 5 ejecuciones con semilla entre 1 y 5 y se calculó la media de tiempos y de desviación.

Algoritmo ES			
Caso	**Coste medio obtenido**	**Desv**	**Tiempo (ms)**
GKD-b_1_n25_m2	0,0000	0,00	No medido
GKD-b_2_n25_m2	0,0000	0,00	No medido
GKD-b_3_n25_m2	0,0000	0,00	No medido
GKD-b_4_n25_m2	0,0000	0,00	No medido
GKD-b_5_n25_m2	0,0000	0,00	No medido
GKD-b_6_n25_m7	36,9192	65,55	3,84E+00
GKD-b_7_n25_m7	32,2670	56,31	2,72E+00
GKD-b_8_n25_m7	24,4014	31,31	4,00E+00
GKD-b_9_n25_m7	45,7637	62,70	3,22E+00
GKD-b_10_n25_m7	35,1925	33,89	3,94E+00
GKD-b_11_n50_m5	3,7085	48,06	1,18E+01
GKD-b_12_n50_m5	7,2844	70,88	1,19E+01
GKD-b_13_n50_m5	8,5195	72,27	8,08E+00
GKD-b_14_n50_m5	9,9233	83,24	7,92E+00
GKD-b_15_n50_m5	4,3659	34,65	7,95E+00
GKD-b_16_n50_m15	139,8630	69,44	1,11E+01
GKD-b_17_n50_m15	88,0914	45,39	1,08E+01
GKD-b_18_n50_m15	107,7540	59,91	7,82E+00
GKD-b_19_n50_m15	99,2251	53,23	1,16E+01
GKD-b_20_n50_m15	55,8000	14,49	1,98E+01
GKD-b_21_n100_m10	49,2106	71,89	2,13E+01
GKD-b_22_n100_m10	34,9562	60,91	2,77E+01
GKD-b_23_n100_m10	37,8170	59,42	2,05E+01
GKD-b_24_n100_m10	32,7662	73,63	1,72E+01
GKD-b_25_n100_m10	40,2160	57,23	1,65E+01
GKD-b_26_n100_m30	318,9830	47,10	3,34E+01
GKD-b_27_n100_m30	295,9270	57,05	3,96E+01
GKD-b_28_n100_m30	358,4960	70,33	2,56E+01
GKD-b_29_n100_m30	309,7230	55,62	4,44E+01
GKD-b_30_n100_m30	234,6390	45,67	3,91E+01
GKD-b_31_n125_m12	22,3617	47,48	3,25E+01
GKD-b_32_n125_m12	49,5857	62,11	2,70E+01
GKD-b_33_n125_m12	44,8375	58,67	3,53E+01
GKD-b_34_n125_m12	43,9128	55,62	3,36E+01
GKD-b_35_n125_m12	48,2032	62,42	4,49E+01
GKD-b_36_n125_m37	419,6500	62,96	7,34E+01
GKD-b_37_n125_m37	471,8540	57,85	2,23E+01
GKD-b_38_n125_m37	490,1090	61,65	6,37E+01
GKD-b_39_n125_m37	528,5900	68,11	2,65E+01
GKD-b_40_n125_m37	409,3480	56,47	4,15E+01
GKD-b_41_n150_m15	40,9578	43,00	6,15E+01
GKD-b_42_n150_m15	109,9870	75,64	3,25E+01
GKD-b_43_n150_m15	72,8826	63,29	4,81E+01
GKD-b_44_n150_m15	73,1569	64,55	4,70E+01
GKD-b_45_n150_m15	64,8298	57,16	3,28E+01
GKD-b_46_n150_m45	458,6410	50,34	4,56E+01
GKD-b_47_n150_m45	382,2820	40,20	9,75E+01
GKD-b_48_n150_m45	508,1380	55,38	4,53E+01
GKD-b_49_n150_m45	501,0280	54,81	4,19E+01
GKD-b_50_n150_m45	761,8830	67,34	3,31E+01

Media Desv: **51,30**
Media Tiempo: **2,60E+01**

Algoritmo BMB			
Caso	Coste medio obtenido	Desv	Tiempo (ms)
GKD-b_1_n25_m2	0,0000	0,00	No medido
GKD-b_2_n25_m2	0,0000	0,00	No medido
GKD-b_3_n25_m2	0,0000	0,00	No medido
GKD-b_4_n25_m2	0,0000	0,00	No medido
GKD-b_5_n25_m2	0,0000	0,00	No medido
GKD-b_6_n25_m7	21,6578	41,28	9,49E+00
GKD-b_7_n25_m7	16,7430	15,79	6,69E+00
GKD-b_8_n25_m7	16,7612	0,00	8,48E+00
GKD-b_9_n25_m7	26,3009	35,10	9,33E+00
GKD-b_10_n25_m7	23,2652	0,00	8,89E+00
GKD-b_11_n50_m5	7,3354	73,74	1,02E+01
GKD-b_12_n50_m5	2,0513	-3,29	1,05E+01
GKD-b_13_n50_m5	9,5894	75,37	1,22E+01
GKD-b_14_n50_m5	2,6613	37,50	1,08E+01
GKD-b_15_n50_m5	2,9438	3,08	1,06E+01
GKD-b_16_n50_m15	62,4366	31,54	4,88E+01
GKD-b_17_n50_m15	88,9011	45,89	4,43E+01
GKD-b_18_n50_m15	54,4863	20,72	4,31E+01
GKD-b_19_n50_m15	93,1396	50,17	4,03E+01
GKD-b_20_n50_m15	47,7151	0,00	4,44E+01
GKD-b_21_n100_m10	20,8901	33,79	5,30E+01
GKD-b_22_n100_m10	31,2433	56,26	4,76E+01
GKD-b_23_n100_m10	14,7113	-4,13	5,75E+01
GKD-b_24_n100_m10	25,5234	66,15	4,46E+01
GKD-b_25_n100_m10	30,8494	44,24	4,17E+01
GKD-b_26_n100_m30	294,1460	42,64	2,69E+02
GKD-b_27_n100_m30	221,8660	42,71	3,06E+02
GKD-b_28_n100_m30	214,4100	50,39	2,92E+02
GKD-b_29_n100_m30	191,7320	28,31	2,50E+02
GKD-b_30_n100_m30	207,8940	38,68	2,92E+02
GKD-b_31_n125_m12	32,9456	64,35	1,09E+02
GKD-b_32_n125_m12	32,3981	42,01	7,66E+01
GKD-b_33_n125_m12	32,4875	42,96	9,62E+01
GKD-b_34_n125_m12	37,0155	47,35	7,99E+01
GKD-b_35_n125_m12	33,2081	45,46	1,01E+02
GKD-b_36_n125_m37	231,8690	32,96	5,01E+02
GKD-b_37_n125_m37	347,4190	42,75	5,06E+02
GKD-b_38_n125_m37	287,9430	34,72	5,95E+02
GKD-b_39_n125_m37	288,1650	41,50	4,62E+02
GKD-b_40_n125_m37	365,9900	51,31	4,58E+02
GKD-b_41_n150_m15	44,8451	47,94	1,29E+02
GKD-b_42_n150_m15	48,0067	44,20	1,54E+02
GKD-b_43_n150_m15	42,1887	36,58	1,51E+02
GKD-b_44_n150_m15	41,3745	37,32	1,32E+02
GKD-b_45_n150_m15	31,8207	12,72	1,44E+02
GKD-b_46_n150_m45	450,7100	49,47	8,68E+02
GKD-b_47_n150_m45	350,3790	34,76	8,78E+02
GKD-b_48_n150_m45	337,5350	32,82	9,15E+02
GKD-b_49_n150_m45	433,4260	47,76	1,00E+03
GKD-b_50_n150_m45	381,0060	34,68	1,11E+03

Media Desv: 32,99
Media Tiempo: 2,09E+02

Algoritmo ILS			
Caso	**Coste medio obtenido**	**Desv**	**Tiempo (ms)**
GKD-b_1_n25_m2	0,0000	0,00	No medido
GKD-b_2_n25_m2	0,0000	0,00	No medido
GKD-b_3_n25_m2	0,0000	0,00	No medido
GKD-b_4_n25_m2	0,0000	0,00	No medido
GKD-b_5_n25_m2	0,0000	0,00	No medido
GKD-b_6_n25_m7	31,2907	59,36	7,58E+00
GKD-b_7_n25_m7	20,9563	32,72	7,63E+00
GKD-b_8_n25_m7	21,8265	23,21	9,45E+00
GKD-b_9_n25_m7	25,6540	33,46	8,17E+00
GKD-b_10_n25_m7	30,8805	24,66	7,63E+00
GKD-b_11_n50_m5	3,9112	50,75	1,16E+01
GKD-b_12_n50_m5	8,4387	74,87	1,05E+01
GKD-b_13_n50_m5	8,2132	71,24	1,03E+01
GKD-b_14_n50_m5	5,1549	67,74	8,36E+00
GKD-b_15_n50_m5	6,6529	57,11	1,30E+01
GKD-b_16_n50_m15	42,7458	0,00	3,93E+01
GKD-b_17_n50_m15	59,9642	19,77	3,81E+01
GKD-b_18_n50_m15	54,7947	21,17	4,40E+01
GKD-b_19_n50_m15	56,8737	18,39	3,54E+01
GKD-b_20_n50_m15	71,3043	33,08	3,78E+01
GKD-b_21_n100_m10	29,5233	53,15	4,80E+01
GKD-b_22_n100_m10	21,6023	36,75	5,12E+01
GKD-b_23_n100_m10	24,5966	37,61	4,46E+01
GKD-b_24_n100_m10	29,1972	70,41	4,21E+01
GKD-b_25_n100_m10	24,0596	28,51	4,51E+01
GKD-b_26_n100_m30	186,7370	9,64	2,50E+02
GKD-b_27_n100_m30	196,4480	35,30	2,46E+02
GKD-b_28_n100_m30	221,5910	51,99	2,69E+02
GKD-b_29_n100_m30	203,1450	32,34	2,51E+02
GKD-b_30_n100_m30	220,3780	42,15	2,38E+02
GKD-b_31_n125_m12	16,1464	27,26	1,05E+02
GKD-b_32_n125_m12	35,2791	46,74	7,14E+01
GKD-b_33_n125_m12	35,0743	47,16	8,17E+01
GKD-b_34_n125_m12	28,9771	32,75	7,88E+01
GKD-b_35_n125_m12	27,4741	34,07	8,21E+01
GKD-b_36_n125_m37	201,6090	22,90	3,76E+02
GKD-b_37_n125_m37	348,1890	42,88	4,90E+02
GKD-b_38_n125_m37	276,9280	32,12	5,00E+02
GKD-b_39_n125_m37	273,3680	38,33	4,63E+02
GKD-b_40_n125_m37	303,5070	41,29	5,06E+02
GKD-b_41_n150_m15	39,9764	41,60	1,18E+02
GKD-b_42_n150_m15	54,6899	51,02	1,13E+02
GKD-b_43_n150_m15	44,5641	39,96	1,21E+02
GKD-b_44_n150_m15	37,1146	30,12	1,30E+02
GKD-b_45_n150_m15	47,7999	41,90	1,06E+02
GKD-b_46_n150_m45	337,4600	32,51	7,91E+02
GKD-b_47_n150_m45	338,0430	32,37	7,06E+02
GKD-b_48_n150_m45	391,1270	42,03	8,28E+02
GKD-b_49_n150_m45	411,7040	45,01	6,78E+02
GKD-b_50_n150_m45	358,7920	30,64	7,44E+02

Media Desv: **34,76**

Media Tiempo: **1,77E+02**

RESULTADOS GLOBALES		
Algoritmo	**Desv**	**Tiempo (ms)**
Greedy	63,99	1,15E+01
BL	54,77	2,25E+01
ES	51,30	2,60E+01
BMB	32,99	2,09E+02
ILS	34,76	1,77E+02

Los algoritmos que mejores resultados han dado son aquellos que usan las trayectorias simples varias veces. Para BMB e ILS ejecutamos varias veces la búsqueda local y nos quedamos con la mejor solución.

En cuanto a BL y ES, ambos obtienen tiempos y resultados muy parecidos. En la Práctica 1, la búsqueda local quedaba rápidamente estancada en un mínimo local y se realizaban pocas evaluaciones. En esta práctica, el ES en las primeras iteraciones (con temperaturas relativamente altas), actualiza más veces la solución que en las últimas (donde la temperatura es más baja). Por tanto, cuando la temperatura es baja, es menos probable que actualicemos la solución, porque, primero, es más difícil que sea mejor (ya llevamos varias iteraciones) y es más difícil tomar una solución peor (por la función de probabilidad). Esto en parte hace que al principio variemos más la solución para evitar mínimos locales pero al final ya no cambia tanto para no producir efectos poco deseados en el fitness que tenemos actualmente. Se ha observado que la práctica totalidad de las ejecuciones de ES terminan porque ha caído en un mínimo local y no obtiene ningún éxito a la hora de actualizar la solución actual.

ES es una opción mucho más deseable que BL ya que obtenemos unos resultados un poco mejores en el mismo tiempo de cálculo y es, en general, más fácil de implementar.