Metaheurísticas(2019-2020)
Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada

Técnicas de Búsqueda basadas en Poblaciones para el Problema de la Máxima Diversidad (MDP)

Alberto Jesús Durán López
DNI: 54142189-M
albduranlopez@gmail.com

Grupo Jueves, 17:30 - 19:30

25 de mayo de 2020

Índice

1. Introducción

Durante el transcurso de la asignatura se ha trabajado con el problema de la máxima diversidad *(Max Diversity Problem)*.
En particular, en esta práctica estudiaremos técnicas de búsqueda basadas en poblaciones para la resolución del problema en cuestión.
Comentaremos todos los pasos y problemas encontrados, detallando minuciosamente todos los detalles y soluciones a los mismos.

Además, se incorporarán tablas para mostrar los resultados de todas las ejecuciones y gráficas para contrastar los modelos (optimización, costes y desviación).

2. Problema de la máxima Diversidad

El problema de la máxima diversidad (*maximum diversity problem*, MDP) es un problema de optimización combinatoria consistente en seleccionar un subconjunto de m elementos ($|M| = m$) de un conjunto inicial N de n elementos (con $n > m$) de forma que se maximice la diversidad entre los elementos escogidos.

El **MDP** se puede formular como:

$$\text{Maximizar: } z_{MS}(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j$$

$$\text{Sujeto a: } \sum_{i=1}^{n} x_i = m \quad \text{con } x_i = \{0,1\},\ i = 1, ..., n \quad \text{donde:}$$

- x es una solución al problema que consiste en un vector binario que indica los m elementos seleccionados.
- d_{ij} es la distancia existente entre los elementos i y j

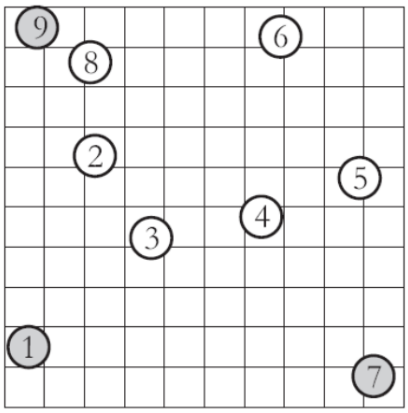


Figura 2.1: MDP-Maximum Diversity Problem

3. Datos y casos considerados

Las ejecuciones se han realizado en un ordenador `Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz, 16GB RAM, 512 SSD`.
Se utilizarán **30 casos** seleccionados de varios de los conjuntos de instancias disponible en la MDPLIB, 10 pertenecientes al grupo `GKD` con distancias Euclideas, n=500 y m=50 *(GKD-c_11_n500_m50 a GKD- c_20_n500_m50)*, 10 del grupo `MDG` con distancias reales en [0,1000], n=500 y m=50 *(MDG-b_1_n500_m50 a MDG-b_10_n500_m50)*; y otras 10 del grupo `MDG` con distancias enteras en 0,10, n=2000 y m=200 *(MDG-a_31_n2000_m200 a MDG- a_40_n2000_m200)*.

Para la realización de la práctica usaremos el lenguaje de programación `C++` ya que se deben probar muchos ejemplos y la ejecución es más rápida al ser un lenguaje de programación compilado.
Todos los archivos se han compilado con la opción de optimización `-O2` y, además, la semilla usada para todas las ejecuciones ha sido 54142189, pudiéndose ésta indicar en la ejecución del archivo en cuestión.

3.1. Esquema de representación de soluciones

Para todos los algoritmos menos para la *Búsqueda Local*, se ha usado una codificación basada en números binarios. Sin embargo, dejaremos la explicación para las siguientes secciones, donde se explicará con detalle en cada algoritmo.

3.2. Descripción de la Función Objetivo

Todos los algoritmos implementados hacen uso de la misma función objetivo, la comentada en el punto (2) anterior y cuyo pseudocódigo mostramos a continuación. Bien es cierto que en el caso de la *Búsqueda Local* se tiene que usar codificación basada en números enteros, sin embargo, dicha función se ve mínimamente modificada.

```
Datos: ContribucionIndep(integer ind, vector sel, matriz distancias)
inicio
    suma ← 0 ;
    para j in sel hacer
        suma ← sel[j] · (distancias[ind][j] + suma) ;
    fin
    devolver suma
fin
```

```
Datos: CosteEstimado(vector sel, matriz distancias)
inicio
    suma ← 0 ;
    para i in |sel| hacer
        si sel[i] = true entonces
            suma ← ContribucionIndep(i, sel, distancias) + suma ;
        fin
    fin
    devolver suma/2
fin
```

Algoritmo 1: Evaluar solución

4. Enfriamiento Simulado

Para el desarrollo del *Enfriamiento Simulado*, dividimos su explicación en distintas componentes:

- `Esquema de representación y solución inicial`: Para este algoritmo se ha usado una codificación basada en números binarios, 1 si se toma el elemento y 0 en caso contrario. Almacenamos la solución en un conjunto *sel* y, nuestro algoritmo que genera las soluciones iniciales, se encargará de crear dichos valores comprobando que se cumplan las restricciones (el número de unos debe ser igual a `m`).

- `Parámetros y esquema de enfriamiento`: La temperatura inicial (T_0) y final (T_f) se calculan de la siguiente forma:

$$T_0 = \frac{\mu \cdot C(s_0)}{-ln(\phi)} \; ; \qquad T_f = 10^{-3}$$

  donde $\phi \in [0, 1]$ es la probabilidad de aceptar una solución un μ por 1 peor que la inicial, cumpliendo $\mu = \phi = 0{,}3$ para nuestro caso.

  $C(S_0)$ es el coste de la solución inicial

  Por otro lado, para el enfriamiento se ha empleado el esquema de Cauchy Modificado, donde β es una constante que calcularemos al principio de nuestro algoritmo y T_{k+1} se calculará en cada iteración.

$$T_{k+1} = \frac{T_k}{1 + \beta \cdot T_k} \quad \text{con} \quad \beta = \frac{T_0 - T_f}{M \cdot T_0 \cdot T_f}$$

  donde M es el número de enfriamientos a realizar.

- `Esquema de generación de vecinos`: El entorno de una solución *sel* está formado por las soluciones accesibles desde ella a través de un movimiento de intercambio. En cada iteración, se aplicará un movimiento de intercambio **Int***(sel,i,j)* que consistirá en intercambiar un elemento i por j en el contenedor de seleccionados(se cambiarán elementos diferentes, 1 por 0). El gran número de intercambios supone un gran número de llamadas a la función objetivo y, en consecuencia, una peor eficiencia. Por ello, realizamos una factorización de la función objetivo, de forma que sólo tengamos que sumar y restar las contribuciones del elemento añadido y quitado, respectivamente.

Mostramos a continuación el pseudocódigo asociado a la función principal de nuestro algoritmo:

```
Datos: ES(integer m, matriz distancias)
inicio
    max_vecinos ← 10 · n ;
    max_exitos ← 0,1 · max_vecinos ;
    evaluaciones ← 100000 ;
    enfriamientos ← 100000 · max_vecinos ;

    sel ← Solucion Aleatoria(n,m) ;
    t_0 ← (μ · Coste(sel))/ − ln(φ) ;
    t_f ← 1/1000 ;
    β ← (t_0 − t_f)/(enfriamientos · t_0 · t_f) ;

    mientras t_f < t_0 and eval < evaluaciones and exitos ≠ 0 hacer
        exitos ← 0 ;
        para i = 0; i < max_vecinos and exitos < max_exitos; i++ hacer
            siguiente ← Intercambio(sel, i, j) ;
            eval++ ;
            dif ← Contribución Nueva - Contribución Vieja ;

            si dif > 0 or U(0,1) < exp(dif/t_0) entonces
                exitos++ ;
                Coste(Siguiente) ← Coste(Siguiente) + dif ;
                sel ← siguiente ;

                si Coste(Siguiente) > Coste(mejor_sol) entonces
                    mejor_sol ← siguiente ;
                fin
            fin
        fin
        t_0 ← t_0/(1 + β · t_0);
    fin
fin
```

Algoritmo 2: Enfriamiento Simulado

5. Búsqueda Local

El algoritmo de *Búsqueda Local* usado para el resto de algoritmos que se explican en las siguientes secciones se corresponde con el implementado en la primera práctica, por ello, conviene recordar como funciona:

- En nuestro problema MDP, el parámetro `i` indica el índice del elemento que se eliminará de la solución y el j por cual se sustituirá. Por tanto, las opciones para `i` serán `m` (los m elementos seleccionados) y las opciones para `j` serán `n-m` (los elementos disponibles en *no seleccionados*), por tanto, el entorno estará formado por `m·(n-m)` elementos.

Datos: EvaluaVecinos
inicio
  $mejora \leftarrow True$;
  $eval \leftarrow 0$;

  mientras *eval<100.000 and mejora* **hacer**
    . . .

    $salir \leftarrow$ False, $c \leftarrow 0$;
    mientras $c < n - m$ *and not salir* **hacer**
      Genera elemento válido j ;
      si $Contrib(s_j) > Contrib(s_i)$ **entonces**
        Int(sel, i, j) ;
        $salir \leftarrow$ True;
        $coste \leftarrow coste + contribNueva - contribAntigua$
      fin
      $c \leftarrow c + 1$;
      $eval \leftarrow eval + 1$;
    fin
  fin
fin

Algoritmo 3: EvaluaVecinos

- En cada iteración, podemos calcular la diferencia de costes entre las dos soluciones recalculando todas las distancias de la función objetivo, sin embargo, esto no es necesario ya que como únicamente añadimos y quitamos 1 elemento, basta con sumar y restar la distancia del nuevo y del viejo elemento al resto de elementos seleccionados, respectivamente. Además, combinamos la factorización del coste con el cálculo de la contribución de los elementos para mejorar aún más la eficiencia.

  Si el intercambio es favorable, es decir, si el coste de la nueva solución (sumando las distancias del nuevo elemento y restando las del elemento anterior) es mayor que la anterior, aceptamos el intercambio. En caso contrario, rechazamos.

- Repetiremos este proceso hasta que se realicen N evaluaciones de la función objetivo o cuando no encuentre mejora en el entorno.

6. Búsqueda Multiarranque Básica

El algoritmo de *BMB* es quizás el algoritmo menos laborioso de nuestra práctica. Éste, consistirá en generar 10 soluciones iniciales aleatorias que serán optimizadas con nuestra *Búsqueda Local* implementada en la práctica 1 y descrita en el apartado anterior. `Criterio de parada`: el algoritmo finalizará cuando no encuentre mejora en todo el entorno o cuando se hayan realizado 10.000 evaluaciones de la función objetivo.

Mostramos la función asociada a la generación de soluciones iniciales. Como en nuestro algoritmo simplemente tenemos que aplicar la *Búsqueda Local*, usamos codificación basada en conjuntos enteros.

```
Datos: Solucion Inicial(n, m)
inicio
    vector<int> sel ;
    mientras sel ≠ m hacer
        random ← rand() %n ;
        si random no insertado entonces
            sel ← sel ∪ random ;
        fin
    fin
    devolver sel
fin
```

Algoritmo 4: Solución inicial

Por último, mostramos el pseudocódigo asociado a la función principal. Se realiza un bucle **for** para realizar un total de 10 iteraciones en las que se generarán soluciones aleatorias y se le aplicará *BL*. La solución devuelta será la mejor de todas.

```
Datos: BMB(integer m, matriz distancias)
inicio
    total ← 10 ;
    matriz soluciones(total) ;
    vector mejor ;

     para i=0 to total hacer
        soluciones[i] ← Solución Aleatoria(n,m) ;
        soluciones[i] ← BusquedaLocal(soluciones[i], distancias) ;
        si Coste(soluciones[i]) > Coste(mejor) entonces
            mejor ← soluciones[i] ;
        fin
    fin
fin
```

Algoritmo 5: Búsqueda Multiarranque Básica

7. Búsqueda Local Reiterada (ILS)

Este algoritmo combina Búsqueda *Local* con mutación. En cata iteración se aplicará *BL* y mutación sobre la mejor solución encontrada y, tras un número de iteraciones, se devolverá la mejor solución encontrada en toda la ejecución. Es decir, se sigue el criterio del mejor como aceptación de la *ILS*.

7.1. Generación de Soluciones Iniciales

El esquema de la solución inicial, al igual que la práctica anterior, está caracterizado por la codificación binaria. Cada solución estará formada por valores booleanos 1 (cuando se toma el elemento) o 0 (en caso contrario).
Mostramos a continuación el algoritmo para la generación aleatoria de la solución inicial, teniendo en cuenta que debe cumplir las restricciones del problema MDP, es decir, que el número de 1 es igual a `m`.

Datos: Solucion Inicial(n, m)
inicio
  vector<bool> sel(n) ;
  mientras $total \neq m$ **hacer**
    $i \leftarrow rand()\,\%n$;
    $sel[i] \leftarrow true$;
    $total \leftarrow n^o$ *de True en sel* ;
  fin
  devolver *sel*
fin

Algoritmo 6: Solución inicial

7.2. Esquema de Búsqueda ILS, ILS-ES

El operador de mutación de *ILS* se basa en un operador de vecino para representación de orden que provoca un cambio brusco en la solución actual. Dicho cambio, será mayor que el considerado en la *BL*, realizándose un total de $m \cdot 0{,}1$ mutaciones.

Datos: Operador Mutación(sel, m)
inicio
  $total \leftarrow m{\cdot}0.1$;
  mientras $total > 0$ **hacer**
    $i \leftarrow$ *Generar índice* ;
    hacer{
      $j \leftarrow$ *Generar índice* ;
    }**mientras**$(sel[i] = sel[j])$;
    Intercambiar sel[i] por sel[j];
    total - - ;
  fin
  devolver *sel*
fin

Algoritmo 7: Operador de Mutación

En total, se han implementado 2 versiones, ambas caracterizadas por realizar 10 iteraciones donde se aplicará 10 veces el algoritmo de *BL* (la primera sobre una solución inicial aleatoria y las 9 restantes sobre mutaciones mutadas).

- `ILS`: Para el algoritmo de *Búsqueda Local* se ha usado el implementado en la práctica 1 y descrito anteriormente.
- `ILS-ES`: Tiene la misma composición que el *ILS* estándar, con la única diferencia que el algoritmo de *Búsqueda Local* considerado es el *ES* implementado en la primera parte de la práctica.

EL pseudocódigo para ambas versiones es el siguiente:

Datos: ILS(*integer m, matriz distancias)*
inicio
  totalBL ← *10* ;
  inicial ← **Solucion Inicial**(*n,m*) ;
  sel ← **Busqueda Local**(*inicial*) ;
  totalBL - - ;

  mientras *totalBL* > 0 **hacer**
    sel ← **OperadorMutacion**(*sel,m*);
    sel ← **Busqueda Local**(*sel, distancias*) ;
    totalBL - - ;
    si *Coste*(*sel*) > *Coste*(*mejor*) **entonces**
      mejor ← *sel* ;
    fin
    en otro caso
      sel ← *mejor* ;
    fin
  fin
fin

Algoritmo 8: Búsqueda Local Reiterada

8. Resultados Obtenidos

Instancia	Mejor Coste	ES	Desv	Time	BMB	Desv	Time
GKD-c_11_n500_m50	19587,13	19581,2	0,03	0,03	19586	0,01	0,03
GKD-c_12_n500_m50	19360,24	19357	0,02	0,04	19360,2	0,00	0,05
GKD-c_13_n500_m50	19366,70	19366,7	0,00	0,03	19366,7	0,00	0,04
GKD-c_14_n500_m50	19458,56	19453,1	0,03	0,04	19458,6	0,00	0,04
GKD-c_15_n500_m50	19422,15	19418,6	0,02	0,03	19421,6	0,00	0,03
GKD-c_16_n500_m50	19680,21	19665,2	0,08	0,02	19680,2	0,00	0,04
GKD-c_17_n500_m50	19331,39	19327,3	0,02	0,03	19331,4	0,00	0,03
GKD-c_18_n500_m50	19461,39	19459,8	0,01	0,03	19461,4	0,00	0,03
GKD-c_19_n500_m50	19477,39	19471,6	0,03	0,04	19477,3	0,00	0,04
GKD-c_20_n500_m50	19604,84	19601,8	0,02	0,05	19604,8	0,00	0,04
MDG-b_1_n500_m50	778030,62	762432	2,00	0,07	768472	1,23	0,04
MDG-b_2_n500_m50	779963,69	764288	2,01	0,05	771856	1,04	0,04
MDG-b_3_n500_m50	776768,44	763946	1,65	0,05	767181	1,23	0,04
MDG-b_4_n500_m50	775394,62	766680	1,12	0,06	765863	1,23	0,04
MDG-b_5_n500_m50	775611,06	753849	2,81	0,02	770415	0,67	0,04
MDG-b_6_n500_m50	775153,69	759891	1,97	0,05	765810	1,21	0,04
MDG-b_7_n500_m50	777232,87	762001	1,96	0,04	766665	1,36	0,04
MDG-b_8_n500_m50	779168,75	764350	1,90	0,05	773351	0,75	0,04
MDG-b_9_n500_m50	774802,19	756551	2,36	0,03	766528	1,07	0,03
MDG-b_10_n500_m50	774961,31	759213	2,03	0,05	771090	0,50	0,04
MDG-a_31_n2000_m200	114139	112544	1,40	0,41	113028	0,97	0,67
MDG-a_32_n2000_m200	114092	112382	1,50	0,50	112651	1,26	0,65
MDG-a_33_n2000_m200	114124	112426	1,49	0,49	112944	1,03	0,65
MDG-a_34_n2000_m200	114203	112573	1,43	0,43	112835	1,20	0,63
MDG-a_35_n2000_m200	114180	112820	1,19	0,44	112944	1,08	0,64
MDG-a_36_n2000_m200	114252	112386	1,63	0,47	112987	1,11	0,64
MDG-a_37_n2000_m200	114213	112244	1,72	0,43	112874	1,17	0,65
MDG-a_38_n2000_m200	114378	112921	1,27	0,40	112991	1,21	0,65
MDG-a_39_n2000_m200	114201	112209	1,74	0,41	112908	1,13	0,64
MDG-a_40_n2000_m200	114191	112819	1,20	0,40	112834	1,19	0,65

Tabla 8.1: Resultados ES/BMB

Instancia	Mejor Coste	ILS	Desv	Time	ILS-ES	Desv	Time
GKD-c_11_n500_m50	19587,13	19587,1	0,00	0,03	19584,6	0,01	0,31
GKD-c_12_n500_m50	19360,24	19360,2	0,00	0,04	19360,2	0,00	0,32
GKD-c_13_n500_m50	19366,70	19366,7	0,00	0,05	19366,7	0,00	0,32
GKD-c_14_n500_m50	19458,56	19458,6	0,00	0,03	19458,6	0,00	0,29
GKD-c_15_n500_m50	19422,15	19421,6	0,00	0,03	19422,1	0,00	0,33
GKD-c_16_n500_m50	19680,21	19680,2	0,00	0,03	19680,2	0,00	0,32
GKD-c_17_n500_m50	19331,39	19331,4	0,00	0,04	19329,4	0,01	0,29
GKD-c_18_n500_m50	19461,39	19461,4	0,00	0,04	19461,4	0,00	0,35
GKD-c_19_n500_m50	19477,39	19477,3	0,00	0,03	19477,3	0,00	0,32
GKD-c_20_n500_m50	19604,84	19604,8	0,00	0,04	19604,8	0,00	0,34
MDG-b_1_n500_m50	778030,62	768893	1,17	0,04	770455	0,97	0,59
MDG-b_2_n500_m50	779963,69	776145	0,49	0,04	768793	1,43	0,38
MDG-b_3_n500_m50	776768,44	769205	0,97	0,05	768283	1,09	0,65
MDG-b_4_n500_m50	775394,62	769641	0,74	0,04	767491	1,02	0,45
MDG-b_5_n500_m50	775611,06	769000	0,85	0,03	765396	1,32	0,38
MDG-b_6_n500_m50	775153,69	759684	2,00	0,04	768544	0,85	0,47
MDG-b_7_n500_m50	777232,87	765951	1,45	0,06	769574	0,99	0,50
MDG-b_8_n500_m50	779168,75	775018	0,53	0,04	771452	0,99	0,54
MDG-b_9_n500_m50	774802,19	770633	0,54	0,05	769227	0,72	0,50
MDG-b_10_n500_m50	774961,31	766173	1,13	0,05	769320	0,73	0,45
MDG-a_31_n2000_m200	114139	113825	0,28	0,41	113008	0,99	4,21
MDG-a_32_n2000_m200	114092	113268	0,72	0,46	112858	1,08	4,23
MDG-a_33_n2000_m200	114124	113223	0,79	0,43	113062	0,93	4,70
MDG-a_34_n2000_m200	114203	113306	0,79	0,38	112887	1,15	4,55
MDG-a_35_n2000_m200	114180	113838	0,30	0,37	112966	1,06	4,87
MDG-a_36_n2000_m200	114252	113220	0,90	0,39	113002	1,09	4,81
MDG-a_37_n2000_m200	114213	113447	0,67	0,38	112946	1,11	4,39
MDG-a_38_n2000_m200	114378	113427	0,83	0,37	113016	1,19	4,62
MDG-a_39_n2000_m200	114201	113589	0,54	0,38	112761	1,26	4,34
MDG-a_40_n2000_m200	114191	113398	0,69	0,39	112959	1,08	4,61

Tabla 8.2: Resultados ILS

Algoritmo	*Desv*	*Tiempo*
Greedy	$1,36$	$0,46$
BL	$1,07$	0,49
ES	$1,15$	$0,17$
BMB	$0,72$	$0,24$
ILS	$0,55$	$0,16$
ILS-ES	$0,70$	$1,78$

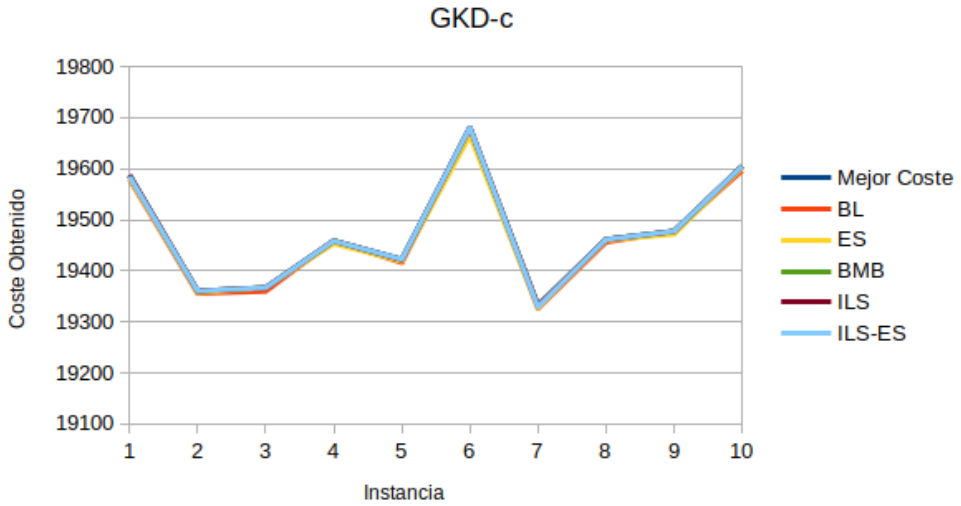
9. Análisis de Resultados

- Observando la tabla de resultados globales 8 anterior, el algoritmo *Greedy* es con el que peor resultados se obtienen. El resto de algoritmos considerados tienen una desviación que se encuentra en un rango de valores muy óptimo y ajustado.
- Por otro lado, todos los algoritmos son muy eficientes, con un tiempo de ejecución de media por debajo de 1 segundo, a excepción de *ILS-ES* que casi dobla este valor.
- Dejando el *Greedy* de lado, obtenemos dos grupos de algoritmos, los que están basados en trayectorias simples (*BL y ES*) y los que están basados en trayectorias múltiples (*BMB, ILS e ILS-ES*), caracterizados por ejecutar las técnicas de trayectorias simples varias veces.

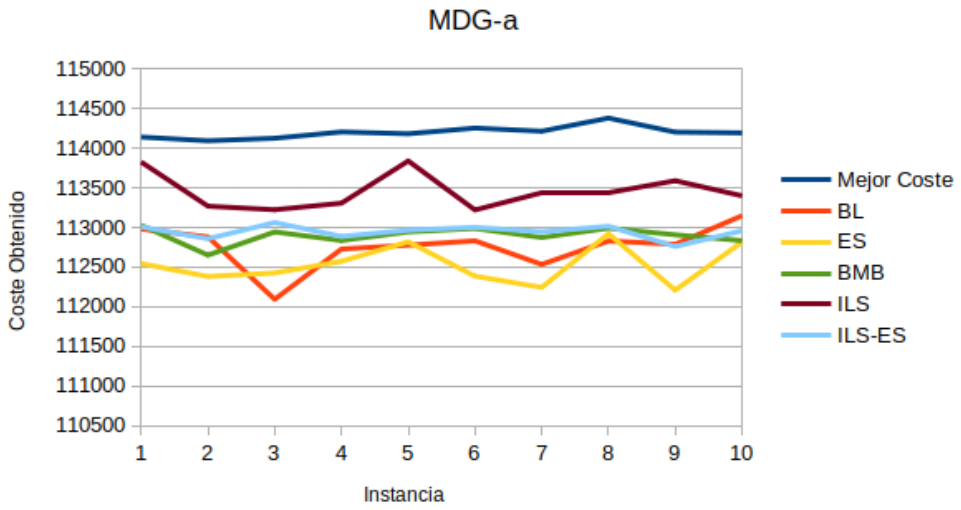
  Teniendo en cuenta este punto, los resultados obtenidos cuadran con el razonamiento teórico, ya que *BMB, ILS e ILS-ES* reducen en torno al 40 % la desviación obtenida de las técnicas basadas en trayectorias simples.

Recordemos los casos seleccionados para la ejecución de los algoritmos: Mostramos los resultados obtenidos en gráficas diferentes según el grupo al que pertenezca cada instancia para reflejar mejor las diferencias:

- `GKD-c`: Conjunto de datos, referentes al grupo de distancias Euclideas con $n = 500$, $m = 50$, en el que mejor resultados se han obtenido, tanto los algoritmos basados en técnicas de trayectorias simples como múltiples llegan al mejor coste. Hay convergencia con todos algoritmos en todas las instancias, de hecho, se aprecia que todas las líneas están superpuestas, obteniendo prácticamente una desviación nula.

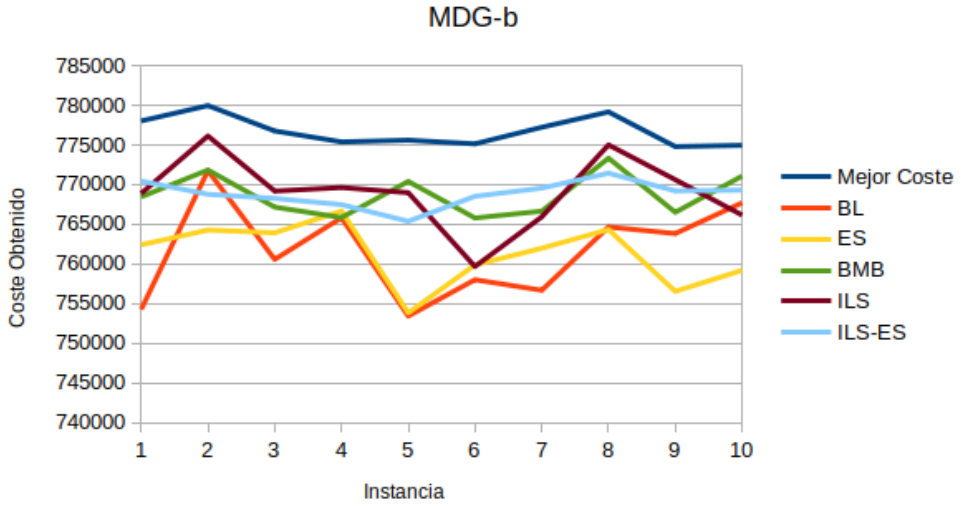


- `MDG-a`: Conjunto de datos correspondiente al grupo de distancias enteras y tamaño de $n = 2000$ y $m = 200$. En todas las instancias los costes obtenidos por los algoritmos basados en técnicas de trayectorias múltiples superan a los que están basados en técnicas de trayectorias simples. Se aprecia como la línea de *ILS* se encuentra por encima del resto, mientras la roja y amarilla, pertenecientes a los algoritmos *BL y ES*, se encuentran por debajo.



- `MDG-b`: Conjunto de datos correspondiente al grupo de distancias reales y tamaño $n = 500$ y $m = 50$. Al igual que en el grupo de instancias anterior, los algoritmos basados en trayectorias simples (BL y ES) se encuentran por debajo del resto.

  Si observamos el trazado por cada uno de los algoritmos, observamos que se presentan altibajos o fluctuaciones en las mismas instancias. Esto es debido a la presencia de óptimos locales, haciendo costoso que nuestros algoritmos puedan salir de estos.

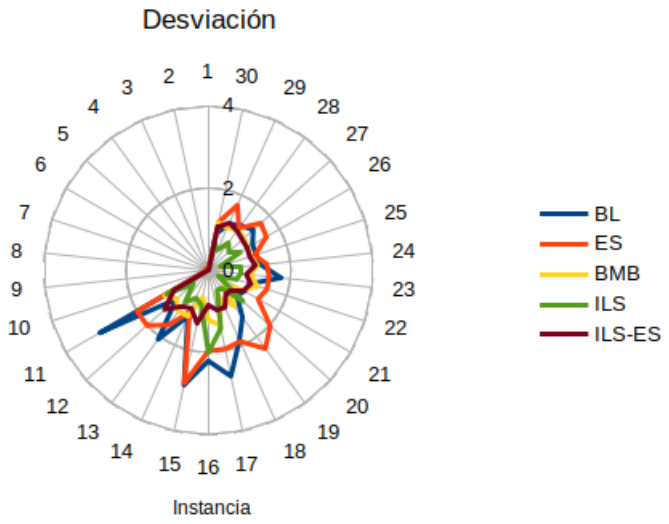


Un aspecto diferente en los algoritmos implementados radica en el número de soluciones usadas en el proceso de optimización. *Enfriamiento Simulado* o *Búsqueda Local* utilizan una sola solución en el proceso de búsqueda.

Por otro lado, técnicas como *Búsqueda Multiarranque Básica* o *Búsqueda Local Reiterada* optimizan un conjunto de soluciones iniciales creadas de forma aleatoria por lo que se obtienen soluciones de más calidad debido a la ventaja que supone explorar diferentes áreas del espacio de búsqueda.
Debido a la posible presencia de óptimos locales, algoritmos como *Búsqueda Local Reiterada* (*ILS*) presentan un factor de mutación brusco, mejorando en gran medida los resultados obtenidos.

9.1. Desviación

Por último, reflejados en el siguiente gráfico en red, mostramos la desviación obtenida de todas las instancia tras la ejecución de cada algoritmo. Como bien sabemos, cuanto menor sea la desviación, mejor será el coste obtenido con nuestro algoritmo. Por tanto, el algoritmo que presente un mayor área bajo su trazado, será peor.



En resumen, podemos considerar que tanto la eficacia como la eficiencia de todos nuestros algoritmos es muy buena por tener una desviación y un tiempo muy pequeños.
Con los que mejor resultados se obtienen son con los algoritmos basados en técnicas de trayectorias múltiples, especialmente el algoritmo de *Búsqueda Local Reiterada* (*ILS*) ya que en media obtiene soluciones más cercanas al mejor valor conocido.

Referencias

-Seminario 1
-Seminario 2
-Seminario 3
-Seminario 4
-Guión de prácticas