METAHEURÍSTICAS-2015/2016

****

PRÁCTICA 2.

Búsquedas Multiarranque para el Problema de la Asignación Cuadrática

****

ALGORITMOS:

- Búsqueda Multiarranque Básica (BMB)
- GRASP
- Búsqueda Local Reiterada (ILS).

Autora: Cristina Zuheros Montes-50616450

- Correo: zuhe18@gmail.com
- Github: https://github.com/cristinazuhe

Horario prácticas: Viernes 17:30-19:30

Fecha: 07 Mayo 2016

ÍNDICE:

1. Descripción del problema.

EL problema de la asignación cuadrática (QAP) que vamos a analizar, trata de conseguir una asignación óptima de n unidades en n localizaciones, conociendo la distancia entre las unidades y el flujo entre las localizaciones. Tanto las distancias entre las unidades como el flujo entre las localizaciones vendrán dados por matrices, que denotaremos como "d" y "f" respectivamente. Así pues, f_{ij} detona el flujo existente entre la unidad "i" y la unidad "j". Análogo para la matriz de distancias.

Una vez tengamos las asociaciones de unidades-localidades, vamos a obtener su coste. El objetivo es conseguir el mínimo coste posible, que nos dará la solución más cercana a la óptima. La función de coste viene determinada por:

coste solucion: $$\min_{\pi \in \Pi_N} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} \cdot d_{\pi(i)\pi(j)} \right)$$

donde π es una solución al problema que consiste en una permutación que representa la asignación de la unidad "i" a la localización "π (i)"

<u>Ejemplo:</u>
Para hacernos una idea más clara, supongamos que tenemos las unidades 0,1,2 y las localidades 0,1,2.
Si la solución encontrada es la permutación {2 0 1} quiere decir que a la unidad 0 le corresponde la localización 2, a la unidad 1 le corresponde la localización 0 y a la unidad 2, la localización 1.

2. Descripción de aplicación de los Algoritmos.

●Greedy.

Mediante este algoritmo asignamos las unidades que tengan mayor flujo a localizaciones que se encuentren céntricas.
La idea es obtener vectores de flujo y de distancias como marginalizaciones de las correspondientes matrices. A partir de dichos vectores, iremos asociando la unidad libre con mayor flujo a la localización libre con menor distancia.

●Búsqueda Local.

Partiremos de una solución inicial, en la que permutaremos dos unidades-localizaciones (solución vecina). Si esta nueva solución mejora a la solución anterior, se aplicará el movimiento y se tomará dicha solución como la actual.

Para ver si esta nueva solución mejora a la solución anterior, basta con obtener el coste del intercambio. Para ello calcularemos:

$$\sum_{k=1,k\neq r,s}^{n}\left[\begin{array}{l} f_{rk}\cdot(d_{\pi(s)\pi(k)}-d_{\pi(r)\pi(k)})+f_{sk}\cdot(d_{\pi(r)\pi(k)}-d_{\pi(s)\pi(k)})+ \\ f_{kr}\cdot(d_{\pi(k)\pi(s)}-d_{\pi(k)\pi(r)})+f_{ks}\cdot(d_{\pi(k)\pi(r)}-d_{\pi(k)\pi(s)}) \end{array}\right]$$

nuevas

viejas

Para obtener la lista de candidatos usaremos la técnica don't look bits.
Finalmente el algoritmo se detendrá cuando se haya explorado el vecindario sin encontrar mejoras o se hayan evaluado 25000 soluciones distintas.

- Pseudocódigo búsqueda local:

Inicializamos el vector dlb a 0's. Inicializo contador a 0.
Mientras que se mejore solución o contador<2500, hacemos dontlookbits:

mejora<-false
for i=0...n-1 && !mejora
si dlb[i]==0
mejora <-false
for j=0...n-1
si i!=j && ***mejora_permutar(i,j)***
mejora <- true
marco unidades i y j en dlb a 0
si !mejora
marco unidad i en dlb a 1

Pseudocódigo funciones auxiliares:

Mejora_permutar(i,j):

mejora <- false
costecambio <- ***coste_Posicion*** *de permutar unidades i,j*
Si costecambio negativo
permuto unidades i y j en solución actual
actualizo coste de solución actual
mejora <-true

coste_Posicion(sol, r, s):

Obtengo el coste de permutar las unidades r, s de solución usando la formula vista en la página 4.
Inicizalizo suma a 0
for k=0...n-1
si k != r y k != s
suma += (m1[r][k]((m2[sol[s]][sol[k]])-(m2[sol[r]][sol[k]]))) +*
(m1[s][k]((m2[sol[r]][sol[k]])-(m2[sol[s]][sol[k]]))) +*
(m1[k][r]((m2[sol[k]][sol[s]])-(m2[sol[k]][sol[r]]))) +*
(m1[k][s]((m2[sol[k]][sol[r]])-(m2[sol[k]][sol[s]])));*

●Funciones auxiliares.

Vamos a ver varias funciones auxiliares que aplicaremos en los algoritmos (mat1 representa flujos y mat2 representa distancias).

- Pseudocódigo de la función coste.

Vemos el pseudocódigo de la función coste vista en el primer apartado que llamábamos descripción del problema:

Coste_Solución:

```
for i=0...n-1
        for j=0...n-1
                si i!=j
                coste += mat1(i,j)*mat2(sol(i),sol(j))
```

- Pseudocódigo de generación de soluciones aleatorias.

También tendremos una función auxiliar para generar una solución aleatoria inicial para los algoritmos. Para ello haremos uso de una misma semilla para todos los algoritmos, que nos irá generando números aleatorios.

Permutar_Solucion_n:

```
for i=0...n-1
        gen1 <- aleatorio entre [0,n-1]
        gen2 <- aleatorio entre [0,n-1] != gen1
        permuto en la solución los indices gen1 y gen2
```

- Nota:

Volvemos a insistir en la representación de soluciones que vamos a emplear:

π hace referencia a una solución al problema que consiste en una permutación que representa la asignación de la unidad "i" a la localización "π (i)" (Ver ejemplo comentado en el primer apartado donde explicamos la descripción del problema).

●Búsqueda Multiarranque Básica (BMB).

Con dicho algoritmo ejecutamos un número de veces (en concreto 25 veces) la búsqueda local que hemos visto anteriormente partiendo de distintas soluciones iniciales aleatorias. De este modo vamos a conseguir bastante diversidad. Nos quedaremos con la mejor solución encontrada.

●GRAPS.

En este algoritmo tratamos de generar una solución greedy probabilística a la que le aplicaremos búsqueda local. Haremos este proceso 25 veces, quedándonos con la mejor solución encontrada.
La generación de la solución greedy probabilística se realizará en dos etapas:

Etapa 1:

Obtengo lista de cantidatos de unidades y localizaciones. Me quedo con dos aleatorias (u1,u2,l1,l2) y las asocio u1→ l1 y u2→ l2

Ejemplo:

Partimos de la solución 13 5 11 3 16 19 14 15 17 1 4 2 18 9 10 12 6 7 8 0.
Obtenemos u1=1; l1=14; u2= 13; l2= 6.
Nos quedaría como solución de la etapa: -1 14 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 6 -1 -1 -1 -1 -1 -1

Etapa 2:

Vamos asignando una unidad-localidad en cada paso, conforme al coste de añadir dicha asignación. Proceso con n-2 pasos pues ya tenemos dos asignaciones realizadas de la etapa anterior.

●Búsqueda Local Reiterada (ILS).

Partiremos de una solución inicial aleatoria a la que le aplicamos búsqueda local. Nos quedaremos con la mejor entre esta y la que teníamos y a ella le aplicaremos mutación. Nos iremos quedando con la mejor en cada iteración (hacemos un total de 25 iteraciones).

3. Descripción en pseudocódigo de los Algoritmos.

●Búsqueda Multiarranque Básica (BMB).

Inicializo:

solucion_aux <- solucion_actual

mejor_coste <- **Coste_Solución**(solucion_aux)

Bucle principal:

Mientras (numero_ejecuciones < 26)

Permutar_Solucion_n*(solucion_aux)*

aplico busqueda_local a solucion_aux obteniendo coste

si coste < mejor_coste

mejor_coste <- coste

solucion_actual <- solucion_aux

Calculo el coste de la solución.

●GRAPS.

Inicializo:

solucion_aux <- solucion_actual

mejor_coste <- **Coste_Solución**(solucion_aux)

Bucle principal:

Mientras (numero_ejecuciones < 26)

Etapa1_SGP*(solucion_aux, n, mat1,mat2)*

aplico busqueda_local a solucion_aux obteniendo coste

si coste < mejor_coste

mejor_coste <- coste

solucion_actual <- solucion_aux

sino solucion_aux <- solucion_actual

Calculo el coste de la solución.

Pseudocódigo funciones auxiliares:

Etapa1(solucion, n, mat1,mat2):
Calculo los potenciales de flujo y distancia
Establezco solucion a -1's
Obtengo mayor y menor potencial de unidades y localizaciones.
Obtengo umbral de ambos potenciales:
    umbral_unidades= mayor_u -(0.3(mayor_u -menor_u))*
    umbral_localizaciones=menor_l +(0.3(mayor_l - menor_l))*
Obtengo lista de unidades y localidades candidatas que superen los umbrales.
Obtengo dos unidades y dos localidades aleatorias de las candidatos. (*)
Asocio unidad1 a localidad1 y unidad2 a localidad2 en solucion
Etapa2*(solucion,n, mat1,mat, u1,l1,u2,l2)*

(*) En el caso en que la lista candidata de unidades solo tenga una unidad, vamos a coger como segunda unidad la siguiente mejor unidad distinta a la primera, aunque no supere el umbral. Lo mismo ocurre para la lista candidata de localizaciones.

Etapa2(solucion,n, mat1,mat, u1,l1,u2,l2)
Creo una matriz donde almaceno el coste de asignar la unidad i,j
for i=0...n-3
    Obtengo coste mínimo y máximo de la matriz
    *nuevo_umbral = c_mínimo + (0,3 * (c_máximo - c_mínimo))*
    Obtengo lista con las asignaciones que superan el umbral.
    Obtengo aleatorio de la lista y realizo la asignacion a solucion.

●Búsqueda Local Reiterada (ILS).

Inicializo:
solucion_aux <- solucion_actual
mejor_coste <- **Coste_Solución**(solucion_aux)

Bucle principal:
Mientras (numero_ejecuciones < 26)
si primera ejecucion
Permutar_Solucion_n*(solucion_aux)*
mejor_coste <- coste
solucion_actual <- solucion_aux
sino
AplicarMutacion*(solucion_aux, n)*
aplico busqueda_local a solucion_aux obteniendo coste
si coste < mejor_coste
mejor_coste <- coste
solucion_actual <- solucion_aux
sino solucion_aux <- solucion_actual

Calculo el coste de la solución.

Pseudocódigo función auxiliar:
AplicarMutacion*(solucion_aux, n)*
Genero aleatorio de 0...n-1
si n ∈ 2,3,4
Genero otro aleatorio al anterior
Intercambio la posición de los dos aleatorios en solución.
sino
Obtener sublista de tamaño n/4 a partir del aleatorio
Permutar_Solucion_n*(solucion_sublista)*
Meter en solucion la permutación realizada.

4. Experimentos y análisis de resultados

Para comentar los resultados obtenidos por los nuevos algoritmos, es interesante que veamos los resultados que obtuvimos para búsqueda local en la práctica anterior (con la misma semilla 550055).

Algoritmo BL			
Caso	**Coste obtenido**	**Desv**	**Tiempo**
Chr20a	3092	41,06	0.000875
Chr20c	24392	72,48	0.000377
Chr22b	7516	21,34	0.000402
Chr25a	5996	57,96	0.000665
Esc32a	170	30,77	0.001245
Esc64a	118	1,72	0.000064
Esc128	72	12,50	0.064397
Kra32	94900	6,99	0.001598
Lipa90a	363339	0,75	0.028767
Sko42	16386	3,63	0.003826
Sko49	24322	4,00	0.009515
Sko81	93978	3,27	0.037218
Sko90	119344	3,30	0.066224
Sko100f	153576	9,16	0.078974
Tai64c	1873908	0,97	0.007462
Tai80a	14181548	5,05	0.019179
Tai100a	21895998	38,19	0.043253
Tai150b	512318987	15,97	0.376338
Tai256c	45027576	2,69	0.679764
Tho150	8382600	10,00	0.392063

Algoritmo BMB			
Caso	Coste obtenido	Desv	Tiempo
Chr20a	2726	24,36131	0,04315
Chr20c	15770	11,51181	0,04033
Chr22b	6786	9,55764	0,04421
Chr25a	5258	38,51423	0,05383
Esc32a	144	10,76923	0,07596
Esc64a	116	0,00000	0,27815
Esc128	66	3,12500	1,76694
Kra32	91220	2,84104	0,08391
Lipa90a	363125	0,69184	0,85541
Sko42	16096	1,79610	0,16828
Sko49	23902	2,20645	0,24508
Sko81	92594	1,75388	1,14184
Sko90	117480	1,68435	1,63165
Sko100f	151130	7,41981	2,36117
Tai64c	1857646	0,09257	0,27285
Tai80a	14041938	4,02064	0,63888
Tai100a	21680016	36,82792	1,20397
Tai150b	512177625	15,93323	9,01047
Tai256c	44913272	2,42562	16,20652
Tho150	8304526	8,97430	9,28300

Vemos que este algoritmo mejora notablemente los resultados que obtenemos con búsqueda local. Esto era de esperar, ya que lo que hacemos es búsqueda local partiendo desde diferentes soluciones iniciales y nos quedamos con la mejor. Aún así, nos encontramos casos en los que seguimos obteniendo malos resultados como chr25a o tai100a.
En cuanto a los tiempos, siguen siendo muy buenos pues básicamente hacemos 25 veces búsqueda local (los tiempos de BMB rondan un 25% más que los de búsqueda local).

Algoritmo GRAPS			
Caso	Coste obtenido	Desv	Tiempo
Chr20a	2726	24,36	0,04316
Chr20c	15770	11,51	0,04579
Chr22b	6592	6,43	0,05140
Chr25a	5258	38,51	0,06489
Esc32a	136	4,62	0,10226
Esc64a	116	0,00	0,66472
Esc128	66	3,13	7,74464
Kra32	91220	2,84	0,10887
Lipa90a	363125	0,69	2,35757
Sko42	16058	1,56	0,24226
Sko49	23754	1,57	0,39522
Sko81	92594	1,75	2,12479
Sko90	117480	1,68	3,06860
Sko100f	151130	7,42	4,55865
Tai64c	1855928	0,00	0,66721
Tai80a	14001864	3,72	1,58443
Tai100a	21680016	36,83	3,50217
Tai150b	510445685	15,54	20,50988
Tai256c	44913272	2,43	110,70053
Tho150	8304526	8,97	20,28801

Mediante GRAPS conseguimos unos resultados bastante buenos (desde esc32a hasta tai80a aprox.). Los resultados son peores cuando disponemos de muchas o de pocas unidades a asociar. Vemos que para los casos chr25a y tai100a no conseguimos mejorar la solución. También cabe destacar que en muchos casos se llega a alcanzar el óptimo o valores muy cercanos a él.
Por otra parte, vemos que para problemas como tai256c, que dispone de muchas unidades a asociar, el tiempo se incrementa notablemente ya que el algoritmo tiene más complejidad. Sin embargo, para los demás casos obtenemos tiempos aceptables, incluso bastante buenos. Mientras no dispongamos de muchas unidades, la relación tiempo invertido y desviaciones obtenidas es muy buena.

Algoritmo ILS			
Caso	Coste obtenido	Desv	Tiempo
Chr20a	2556	16,61	0,04026
Chr20c	18048	27,62	0,04226
Chr22b	6478	4,59	0,04441
Chr25a	4214	11,01	0,05383
Esc32a	140	7,69	0,07660
Esc64a	116	0,00	0,27536
Esc128	66	3,13	1,78764
Kra32	92120	3,86	0,08365
Lipa90a	363183	0,71	0,87742
Sko42	16182	2,34	0,17125
Sko49	23896	2,18	0,25485
Sko81	92842	2,03	1,12740
Sko90	117590	1,78	1,59676
Sko100f	151172	7,45	2,30540
Tai64c	1855928	0,00	0,27091
Tai80a	13986988	3,61	0,64913
Tai100a	21755410	37,30	1,23061
Tai150b	508165837	15,03	8,94221
Tai256c	44951030	2,51	16,26942
Tho150	8307036	9,01	9,22539

Usando ILS obtenemos desviaciones leves, es decir, buenos resultados en la mayoría de los casos. Destaca tai100a pues obtiene una solución que dista bastante de la óptima aunque su tiempo de ejecución es muy rápido. Por lo general obtiene resultados bastante buenos, llegando en varios casos al óptimo.

Los tiempos son bastante bajos en prácticamente todos los casos. Al tener muchas unidades-localizaciones en los problemas, como tai256c o tho150, obtenemos mayores tiempos pero compensa el hecho de que las desviaciones son leves.

Veamos la comparación entre los algoritmos:

Algoritmo	Desv	Tiempo
Greedy	104,55	0
BMB	9,22535	2,27028
GRAPS	8,67832	8,94125
ILS	7,92206	2,26624

Cabe comentar antes de nada, que para todos estos resultados que mostramos hemos usado la semilla 5500055.

Comparando algoritmos:
Vemos que con el Greedy obtenemos una desviación realmente alta, luego no nos aporta ningún beneficio utilizarlo.

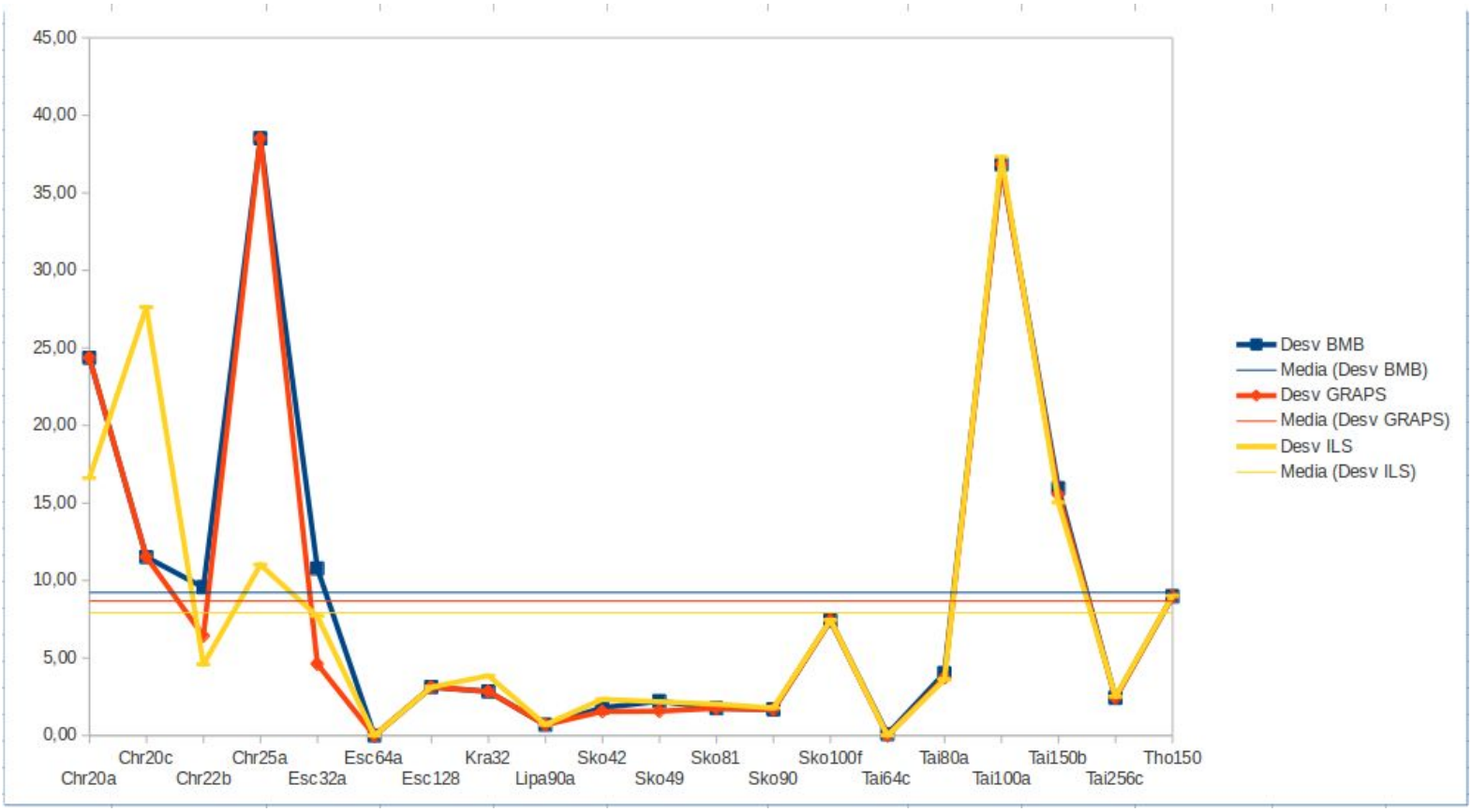
Al aplicar el algoritmo de búsqueda multiarranque básica vemos que la desviación en media es bastante buena. Dicho algoritmo aplica búsqueda local varias veces desde distintas soluciones iniciales. Esto nos permite salir de óptimos locales. El problema es que nos llevará posiblemente a otros óptimos locales (mejores que los anteriores) pero que no llegan a ser óptimos globales y darnos la solución óptima. Sin embargo, es un algoritmo bastante bueno ya que obtiene "leves" desviaciones en poco tiempo.

En cuanto a GRAPS, vemos que mejora a BMB con respecto a las desviaciones obtenidas pero empeora en cuanto al tiempo de ejecución. En comparación con BMB parece ser que va peor, pero no nos podemos dejar engañar, pues no olvidemos que estamos trabajando con una semilla concreta para unos datos concretos. Seguramente para otros datos, otra cantidad en media de unidades y otra semilla que nos inicialice soluciones desde otro punto, podríamos

obtener que GRAPS obtiene mejores resultados que BMB, o viceversa. Siempre considerando la relación entre desviaciones y tiempos. Finalmente nos encontramos con ILS. Parece ser que mejora a todos los algoritmos anteriores en media, tanto en desviaciones como en tiempos. Al igual que comentábamos con GRAPS y BMB, esta mejora no nos permite afirmar que ILS sea mejor algoritmo que los demás. Simplemente, en este experimento consigue mejores resultados, siendo estos muy buenos.

Destaca el hecho de que GRAPS tarde tanto en comparación con los otros algoritmos. Esto que se debe a que la generación de la solución greedy probabilística requiere de bastantes cálculos así como de soluciones y matrices auxiliares que hacen un cómputo más lento.

A continuación vamos a ver un gráfico donde comparamos las desviaciones particulares y medias para los distintos casos de estudio utilizando ILS, GRAPS y BMB:



Vemos que en los problemas con mayor número de unidades-localizaciones, encontramos soluciones con el mismo coste para los tres algoritmos. Esto se puede deber a que no se realicen mejoras al partir de una nueva solución inicial.

Sin embargo en los primeros casos, donde el tamaño de la muestra es menor, vemos que el coste que obtenemos de las soluciones va variando conforme al algoritmo que usamos. No obstante, no podemos decir que un algoritmo sea mejor que otro, pues vemos que las desviaciones van fluctuando entre sí.

También visualizamos las desviaciones medias de los tres algoritmos, confirmando que el que produce menor desviación media es ILS, aunque para algunos casos como chr20c obtiene el peor resultado.

Valorando casos particulares:

Viendo las tablas anteriores, por un lado observamos que el caso thai100a es uno de los más problemáticos en el sentido de que con cualquiera de los algoritmos realizados obtenemos soluciones muy malas. Tendremos que realizar otra serie de algoritmos para conseguir aproximarnos más a la solución óptima. Este caso hace que la desviación en media aumente bastante para todos los algoritmos.

Por otro lado, destaca el caso esc64a con el que obtenemos el resultado óptimo con cualquiera de los algoritmos.

Número de veces que es mejor…:

Vemos que los algoritmos BMB, GRAPS e ILS van mejorando progresivamente las desviaciones pero esto no implica que el último de ellos sea el mejor algoritmo para todos los casos. Hemos recopilado para cada caso cuál es el mejor algoritmo obteniendo dichos resultados:

Nuevo de veces que es mejor:

- Greedy: 0
- BMB: 0
- GRAPS: 12
- ILS: 8

Llama la atención el hecho de que GRAPS obtiene un mayor número de casos en los que obtiene una mejor solución que los otros algoritmos. Sin embargo, no es el algoritmo que obtiene mejor desviación media (era ILS).

Esto se debe a que, aunque ILS no sea la mejor solución en un caso particular, sí que suele ser mejor en media y no presenta fuertes picos (desviaciones elevadas) que incrementen fuertemente la desviación.

Usando distintas semillas:
Finalmente en nuestro análisis hemos probado a ejecutar los algoritmos con distintas semillas. Veamos las desviaciones de, por ejemplo:

- Con semilla 6060:
  BMB: 9.1014
  GRAPS: 7.9457
  ILS: 9.0763

- Con semilla 1111:
  BMB: 8.7918
  GRAPS: 8.5589
  ILS: 10.5476

- Con semilla 9090:
  BMB: 10.0493
  GRAPS: 9.0877
  ILS: 9.4475

Aunque con la semilla 550055, que hemos analizado anteriormente, obtenemos las mejores desviaciones con ILS, vemos que al cambiarla vamos obteniendo mejores desviaciones con GRAPS.
Esto nos vuelve a mostrar que no hay un algoritmo que sea mejor que otro siempre, sino que dependiendo del problema y de las circunstancias iniciales interesará usar uno u otro.

5. Procedimiento para el desarrollo de la práctica

Para la realización de la práctica he hecho uso de los pdfs seminario2b, guión de prácticas y temas de teoría. He seguido los pseudocódigos que vemos en dichos pdfs, teniendo en cuenta las condiciones que se imponen en el guión de prácticas.