Práctica 2

Búsquedas Multiarranque para el
Problema de la Asignación Cuadrática
BMB • GRASP • ILS

Mario Ruiz Calvo
mariorc@correo.ugr.es
33958755-Z

Grupo 2
(J 17:30-19:30)

Curso 2014-2015

INDICE

1. Descripción del Problema

El problema de asignación cuadrática es un problema de optimización combinatioria.. En éste se trata de asignar N unidades a una cantidad N de localizaciones en donde se considera un costo asociado a cada una de las asignaciones. Este costo dependerá de las distancias y flujo entre las instalaciones. Por lo que el problema consiste en encontrar la asignación óptima de caada unidad a una localización. La función objetivo mide la bondad de una asignación, considerada el producto de la distancia entre cada par de localizaciones y el flujo que circula entre cada par de unidades.

2. Componentes de los algoritmos

- Esquema de representación: La solución se representa con un vector en forma de permutación π de tamaño n, donde los indices del vector corresponden a las unidades y el contenido a las distancias.

- Función objetivo:

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

- Generación de soluciones aleatorias:

```
GenerarSolucionAleatoria(sol)
        Desde i=1 hasta n (tamaño del problema)
                Hacer
                        r = rand(0,n)
                Mientras r este contenido en sol
                sol[i] = r
        Fin
Fin
```

4

- Descripción del algoritmo de búsqueda local:

```
BúsquedaLocalPrimeroMejor(sol, coste)
        Hacer
                Hacer
                        espacioCompleto=generarVecino(sol, svec, r, s, reinciar)
                        reiniciar=false
                        costoFactorizado(sol, r, s, coste_vecino)
                        Nevaluaciones = nevaluaciones +1
                Mientras (coste_vecino>=0 && !espacioCompleto)

                Actualizar MejorSolucion (sol, svec, coste, coste_vecino)
                Si coste_vecino<0 reinciar=true

        Mientras (coste_vecino<0 && n_evaluaciones<25000
Fin
```

- Coste factorizado:

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

- Generar vecinos para la búsqueda local: El tamaño del entorno es $\frac{n\cdot(n-1)}{2}$

```
generarVecino(sol,vecino, r, s, reiniciar)

        Static i=0,j=1

        Si reiniciar
                I=0, =01
        Si i==n-1
                Devolver true

        r=i, s=j, vecino=sol

        Intercambiar (vecino[i], vecino[j]

        Si j>n-1
                i=i+1
                j=i+1
        Sino
                j=j+1

        Devolver false
Fin
```

3. Algoritmos BMB, GRASP e ILS

- Algoritmo Búsqueda Multiarranque Básica:

```
BMB(sol, coste)

        GenerarSolucionAleatoria(sol)
        BusquedaLocalPrimeroMejor(sol,coste)

        Desde i=1 hasta 24 hacer
                GenerarSolucionAleatoria(sol_bl)
                BusquedaLocalPrimeroMejor(sol_bl,coste_bl)
                ActualizarMejorSolucion(sol, sol_bl,coste,coste_bl)
        Fin
Fin
```

- Algoritmo GRASP:

```
GRASP(sol, coste)

        greedyAleatorio(sol,0.3,coste)
        BusquedaLocalPrimeroMejor(sol,coste)

        Desde i=1 hasta 24 hacer
                greedyAleatorio(sol,0.3,coste)
                BusquedaLocalPrimeroMejor(sol_bl,coste_bl)
                ActualizarMejorSolucion(sol, sol_bl,coste,coste_bl)
        Fin
Fin
```

6

```
greedyAleatorio(sol,alfa,coste)
        Obtener potenciales flujo y distancia
        Quicksort(potF)
        Quicksort(potD)

        umbral_unidades = max(PotF) – alfa( min(PotF)-max(PotF))
        umbral_localizaciones = max(potD) – alfa( min(potD)-max(potD))

        Mientras (PotF[i] <= umbral_unidades && i<n)
                Insertar PotF[i] ListaCandidatosUnidades
        Mientras (potD[i] <= umbral_localizaciones && i<n)
                Insertar potD[i] ListaCandidatosLocalizaciones
        a=rand(0,size(ListaCandidatosUnidades))
        b=rand(0,size(ListaCandidatosLocalizaciones))
        c=rand(0,size(ListaCandidatosUnidades))  (a distinto de c)
        d=rand(0,size(ListaCandidatosLocalizaciones)) (b distinto de c)
        sol[a]=b
        sol[c]=d

        Desde cont=3 hasta n hacer
                Desde i=1 hasta n hacer
                        Desde k=1 hasta n hacer
                                Si no esta la asignacion sol[i]=k
                                Insertar (i,k) en ListaCandidatos
                                Calcular coste asociado a (i,k) e insertarlo en costesLC
                        Fin
                Fin

                ordenar(costesLC)
                umbral=min(costesLC)+alfa(max(costesLC)-min(costesLC))

                Mientras (ListaCandidatos[i]<=umbral && i<size(costesLC)
                        Insertar i en ListaRestringidaCandidatos

                R= rand(0, size(ListaRestringidaCandidatos))

                sol[LC[LRC[r].first]=LC[LRC[r].second
        Fin
Fin
```

- Algoritmo Búsqueda Local Reiterada:

```
ILS(sol, coste)

        GenerarSolucionAleatoria(sol)
        BusquedaLocalPrimeroMejor(sol,coste)

        Desde i=1 hasta 24 hacer
                MutacionILS(sol,sol_bl)
                BusquedaLocalPrimeroMejor(sol_bl,coste_bl)
                ActualizarMejorSolucion(sol, sol_bl,coste,coste_bl)
        Fin
Fin
```

```
MutacionILS(sol, sol_mutada)
        inicial=rand(0,n-(n/4)-1)
        sol_mutada=sol
        Desde i=1 hasta n/4
                si r=rand(inicial,inicial+(n/4)-1) no pertenece a aux
                        Insertar r en aux
        Desde i=1 hasta n/4
                intercambia(sol_multada[aux[r]], sol_mutada[aux[i+1]]
Fin
```

4. Procedimiento para la práctica

Para el desarrollo de la práctia se ha partido de un código desde cero, siguiendo como referencia el código proporcionado en la plataforma DECSAI y en el pseudocódigo y explicación de los seminarios y las tranparencias de teoría.

La practica se orgainza en las siguientes carpetas:

- bin: contiene los ficheros ejecutables
- data: contiene los ficheros .dat de donde se extraen los flujos y distancias.
- Include: contiene las ficheros .h
- lib: contien la librerias con todos lof ficheros objeto
- obj: contiene los ficheros objeto .o

- resultados: contiene las tablas con los resultados que se han obtenido para los distintos algoritmos y las desviaciones de tiempo y costo.

- Src: contiene el código fuente.

Además el directorio raíz contiene un fichero makefile para compilar los ejecutables y dos scripts con los que se obtienen los resultados en un formato fácil de trasladar a las tablas.

Ejecutando el fichero makefile se obtienen dos ejecutables: main y obtener_resultados:

main: Ejecuta la búsqueda local primer mejor, la búsqueda multiarranque básica y la busqueda local retroactiva, con una semilla fija (5555) , mostrando el costo obtenido y el tiempo.

-modo de empleo (desde directorio practica2):

./bin/main data/nombre_del_fichero.dat

obtener_resultado: Realiza una ejecución de un algoritmo pasado por parámetro con una semilla variable (también pasada como parámetro) e imprime el coste y el tiempo de ejcución en un formato adecuado para la extracción de datos para las 20 instancias.

-modo de empleo (desde directorio practica2):

./bin/obtener_resultado id_algoritmo semilla

identificador de algoritmo:
- 0: greedy
- 1: búsqueda local primero mejor
- 2: BMB
- 3: GRASP
- 4: ILS

Ejecutando el script obtener_resultado.ssh se realizan 20 ejecuciones DE LAS 20 INSTANCIAS del algoritmo pasado por parámetro, cada una con diferente semilla. Se imprimran los resultados en un formato adecuado para recoger los datos en tablas.

-modo de empleo (desde directorio practica2):

./obtener_resultados.ssh id > resultados/nombre

5. Análisis de resultados

Para obtener los resulados se han realizado 20 ejecuciones de las 20 instancias, calculando la media de los costos y los tiempos obtenidos. La semilla inicial ha sido 1111, incrementandose en 1111 en cada ejecución.

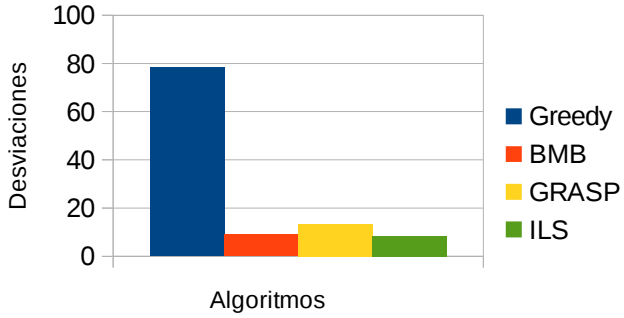
Greedy					
Caso	**Desv**	**Tiempo**	**Caso**	**Desv**	**Tiempo**
Chr20b	342,82	0,000046	**Sko64**	18,39	0,000132
Chr20c	456,63	0,00003	**Sko72**	15,16	0,000161
Chr22a	130,31	0,000045	**Sko81**	15,11	0,000208
Chr22b	134,48	0,000034	**Sko90**	13,49	0,000278
Els19	124,42	0,000027	**Sko100a**	14,37	0,000185
Esc32b	140,48	0,00007	**Sko100b**	13	0,000207
Kra30b	28,04	0,000056	**Sko100c**	12,89	0,00018
Lipa90b	29,36	0,000422	**Sko100d**	14,07	0,000227
Nug30	21,69	0,00002	**Sko100e**	14,64	0,000178
Sko56	16,09	0,000104	**Wil50**	10,59	0,000056

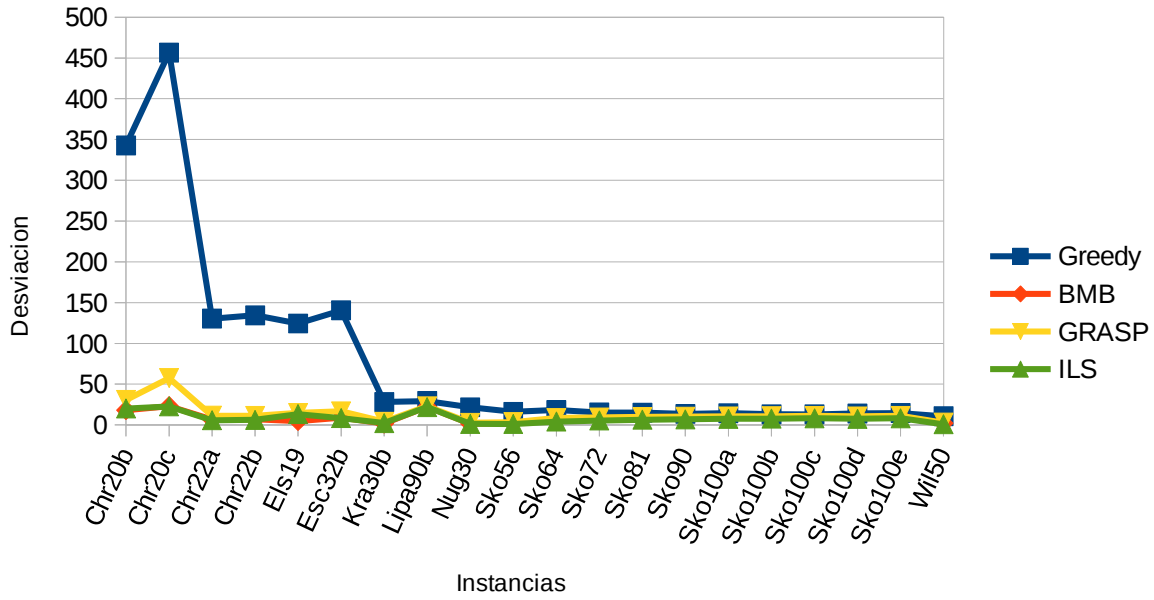
Búsqueda Multiarranque Básica					
Caso	**Desv**	**Tiempo**	**Caso**	**Desv**	**Tiempo**
Chr20b	**18,19**	0,0322762	**Sko64**	7,56	1,76986995
Chr20c	23,02	0,05375345	**Sko72**	7,83	2,0273598
Chr22a	6,76	0,05396085	**Sko81**	8,53	2,3035408
Chr22b	6,78	0,041239	**Sko90**	9,25	2,53211995
Els19	**4,65**	0,07000815	**Sko100a**	9,61	2,81506765
Esc32b	8,93	0,1624712	**Sko100b**	9,70	2,82056755
Kra30b	**1,58**	0,2571565	**Sko100c**	10,14	2,81435345
Lipa90b	22,36	2,5517834	**Sko100d**	9,61	2,81643955
Nug30	1,40	0,32085595	**Sko100e**	10,23	2,8064997
Sko56	2,24	1,56552555	**Wil50**	2,22	1,3774807

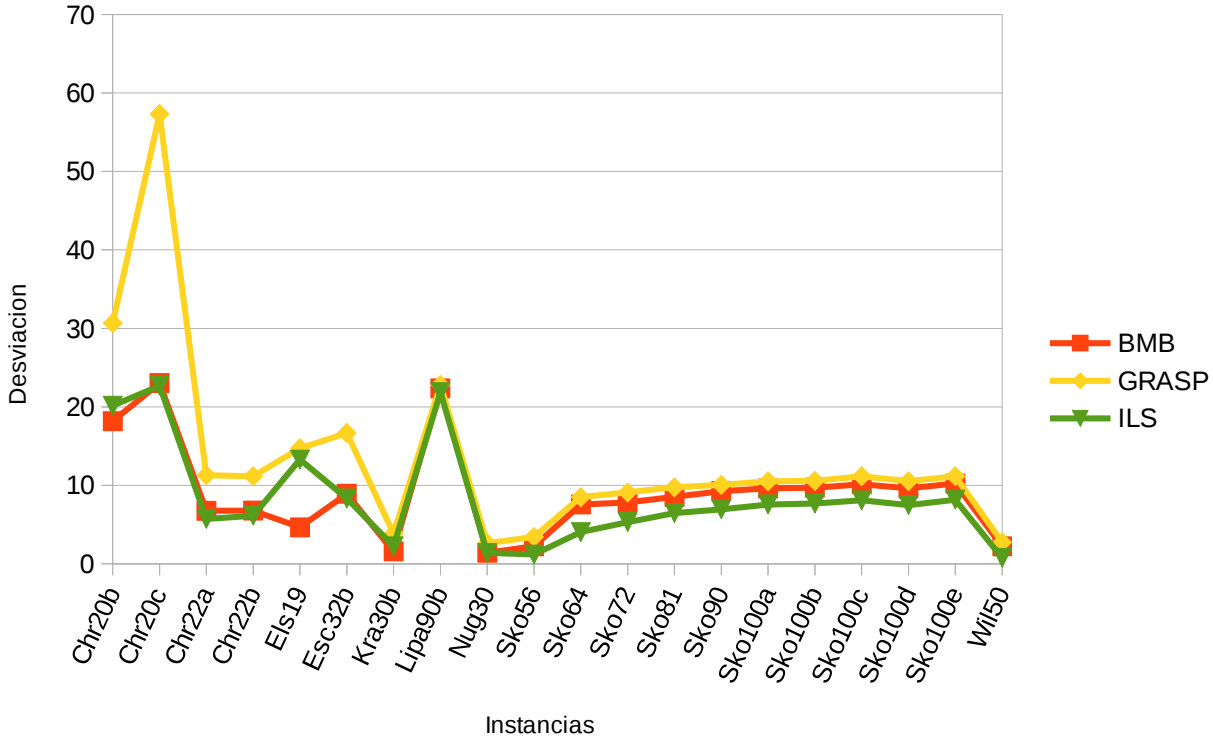
GRASP					
Caso	Desv	Tiempo	Caso	Desv	Tiempo
Chr20b	30,68	0,02254675	**Sko64**	8,49	2,90748115
Chr20c	57,33	0,0201178	**Sko72**	9,1	5,0068252
Chr22a	11,29	0,0266868	**Sko81**	9,76	8,74164605
Chr22b	11,16	0,0245897	**Sko90**	10,05	14,26395585
Els19	14,73	0,01977725	**Sko100a**	10,52	23,60501605
Esc32b	16,67	0,1348181	**Sko100b**	10,57	23,64276455
Kra30b	3,85	0,1146766	**Sko100c**	11,16	23,6250388
Lipa90b	22,81	14,21559575	**Sko100d**	10,54	23,77112655
Nug30	2,63	0,11670675	**Sko100e**	11,17	23,64137775
Sko56	3,43	1,61869895	**Wil50**	2,75	1,0501466

Búsqueda Local Reiterada					
Caso	Desv	Tiempo	Caso	Desv	Tiempo
Chr20b	20,15	0,0207559	**Sko64**	**4,04**	1,8037458
Chr20c	**22,70**	0,02647625	**Sko72**	**5,31**	2,045285
Chr22a	**5,73**	0,0305828	**Sko81**	**6,47**	2,3053308
Chr22b	**6,13**	0,02144895	**Sko90**	**6,94**	2,57434145
Els19	13,28	0,0246226	**Sko100a**	**7,56**	2,84954515
Esc32b	**8,33**	0,0952963	**Sko100b**	**7,67**	2,8497369
Kra30b	2,24	0,08859745	**Sko100c**	**8,09**	2,8254733
Lipa90b	**21,88**	2,56483225	**Sko100d**	**7,5**	2,8301819
Nug30	**1,39**	0,1220447	**Sko100e**	**8,18**	2,82711605
Sko56	**1,18**	1,3010718	**Wil50**	**0,77**	1,1134589

Algoritmo	Desv	Tiempo
Greedy	78,3	0,0001333
BMB	9,03	1,46
GRASP	13,43	8,33
ILS	**8,28**	1,42

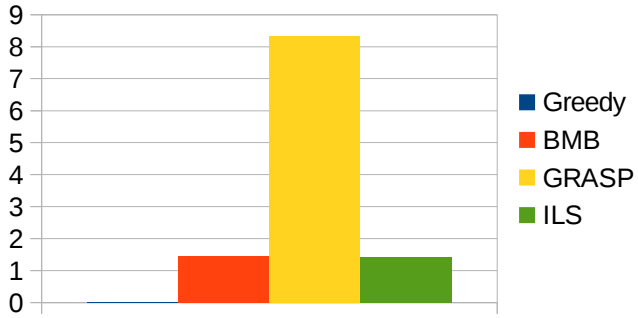


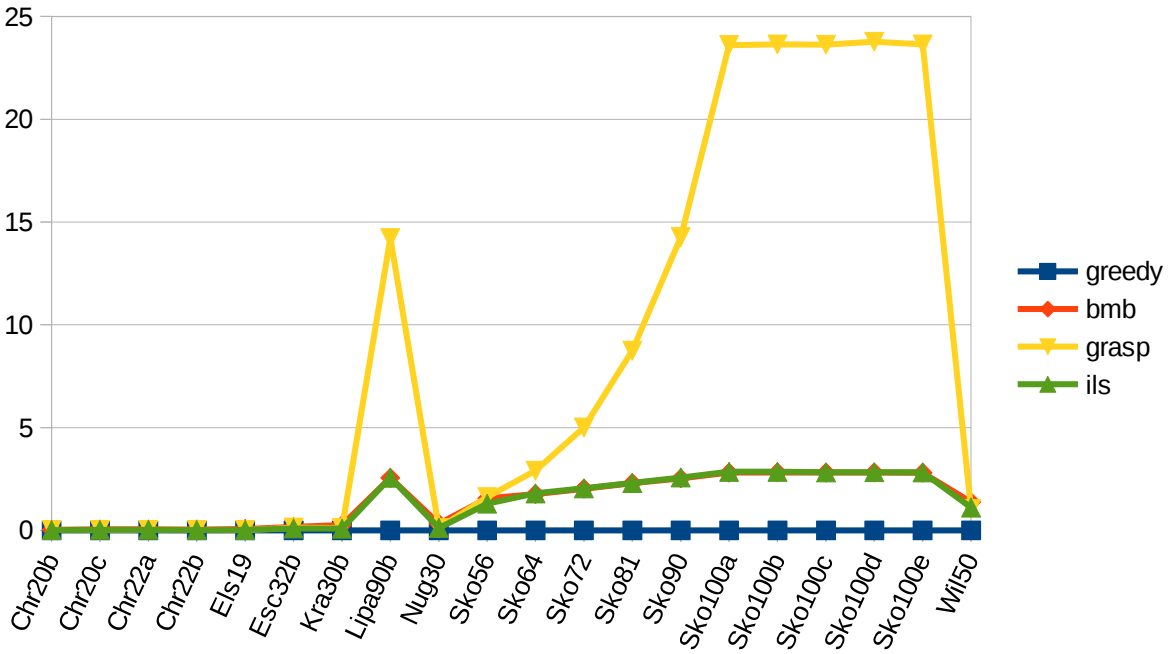




Como es lógico, salvo en algunos casos, el algorimto ILS obtiene mejores costos que BMB pues no parte siempre de una solución aleatoria sino de una mutación de la mejor solución.

Sin embargo por el mismo motivo, es extraño que BMB obtenga mejores resultados que GRASP que hace la búsqueda local a partir de un greedy aleatorizado aunque en la mayoria de los casos se acerca mucho a BMB (sobretodo en tamaños relativamente grandes).





En cuanto a tiempos, GRASP tiene tiempos mucho mayores con tamaños altos, mientras que con tamaños bajos tiene tiempos similares a BMB e ILS. Los tiempos de BMB e ILS son similares en todas las instancias y aumentan tambien con tamaños altos pero sin ser demasiado significativos.