METAHEURÍSTICAS-2015/2016

****

PRÁCTICA 1.

Búsquedas por Trayectorias para el Problema de la Asignación Cuadrática.

****

ALGORITMOS:

- Greedy.
- Búsqueda Local.
- Enfriamiento Simulado.
- Búsqueda Tabú.
- Búsqueda Tabú Extendida.

Autora: Cristina Zuheros Montes-50616450

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

Horario prácticas: Viernes 17:30-19:30

Fecha: 07 Abril 2016

ÍNDICE:

1. Descripción del problema.

EL problema de la asignación cuadrática (QAP) que vamos a trabajar, 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:

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

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.

●Enfriamiento Simulado.

Con dicho algoritmo vamos a permitir ir a soluciones peores, con el fin de poder salir de óptimos locales. La temperatura nos marcará en qué medida pueden ser aceptados vecinos peores. Usaremos la técnica de enfriamiento del esquema de Cauchy modificado.

La temperatura será enfriada cuando finalice la iteración actual: si se han generado un número máximo de vecinos o si se han aceptado un número máximo de vecinos.

Al igual que en Búsqueda Local, iremos generando vecinos intercambiando dos unidades-localidades , esta vez, aleatorias de la solución.

Finalmente, el algoritmo parará al llegar a un número máximo de evaluaciones prefijado (25000) o cuando el número de éxitos en el enfriamiento sea 0.

●Búsqueda Tabú.

Al igual que en Enfriamieto Simulado, vamos a permitir ir a soluciones peores para salir de los óptimos locales.

Para ello vamos generando vecinos aleatorios y dispondremos de una memoria a corto plazo (lista tabú). Los vecinos que vayamos tomando como que mejoran nuestras solución, vamos a ir añadiéndolos a dicha lista.

Estos movimientos serán inválidos durante un cierto tiempo (tenencia tabú) a no ser que cumplan ciertas restricciones de satisfacción (superen nivel de aspiración).

●Búsqueda Tabú Extendida.

Conservamos las ideas de la Búsqueda Tabú vista anteriormente pero añadimos memoria a largo plazo y reinicialización.

Memoria a largo plazo: Crearemos una matriz simétrica que mantendrá el número de veces que se ha producido la asignación i-di en las soluciones aceptadas. Básicamente nos sirve para obtener los movimientos que hemos realizado con mayor frecuencia para que podamos hacer una reinicialización de la solución formada por los movimientos menos usados.

Reinicialización: Tendremos estrategias con probabilidad aleatoria para reinicializar la solución:

0.25 → Generamos solución nueva aleatoria

0.25 → Continuamos con la mejor solución encontrada hasta el momento.

0.5 → Usamos memoria a largo plazo

Además iremos cambiando el tamaño de la lista tabú.

3. Descripción en pseudocódigo de los Algoritmos.

Antes de describir los algoritmos, vamos a ver varias funciones auxiliares que serán comunes para todos ellos.

Calculo el coste de la solución:

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

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
```

● Greedy.

Calculamos los vectores de potenciales:

```
for i=1...n-1
    potencial_f[i]<-0, potencial_d[i]<-0;
    for  j=0...n-1
        potencial_f[i] += mat1[i][j];
        potencial_d[i] += mat2[i][j];
```

Asignamos las unidades-localizaciones

```
for i=0...n-1
     for j=0...n-1
        Obtengo la unidad con mayor flujo que no esté asignada - mayor potencial_f.
        Le asigno la localización con menor distancia que no tenga asociada unidad  - menor potencial_d.
Introduzco en la solución las unidades-localizaciones obtenidas.
```

Calculo el coste de la solución.

●Búsqueda Local.

Inicializamos el vector dlb a 0's.
Mientras que se mejore solución, 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, i, j):
Obtengo el coste de permutar las unidades i, j de solución usando la formula vista en la página 4.

●Enfriamiento Simulado.

Inicializo los parámetros:

*t_inicial = (0.3*costeactual)/(-log(0.3)); t_final = 0.003;*
t_actual=t_inicial
max_vecinos = 10n; max_exitos = 0.1max_vecinos;
n_enfriamientos = 2500/max_vecinos

Bucle principal:

for i=0...n_entriamientos
n_exitos=0
for j=0...max_vecinos
***permutar_Solucion** genera vecino aleatorio con pos1, pos2*
Obtengo diferencia de coste entre soluciones vecina y actual
Si diferencia<0 o U(0,1) < exp(-diferencia/t_actual) - Criterio Metrópolis: n_exitos++
Actualizo coste actual como coste vecino.
Si coste de solución actual < coste solución global
Actualizo solución global como la solución actual
Actualizo coste global como coste actual
Sino, solución vecino vuelve a su estado solución inicial
Si n_exitos >= max_exitos termino bucle
Enfrio_temperatura_actual
si n_exitos=0 termino ejecución

Pseudocódigo funciones auxiliares:

permutar_Solucion

pos1<- elemento aleatorio de la solucion
pos2<- elemento aleatorio de la solucion !=pos1
intercambio pos1 y pos2 en la permutación solucion

Enfrio_temperatura_actual

*Calculo beta = (t_inicial -t_final) / (M*t_inicial*t_final)*
*Devuelvo nueva temperatura = t_actual/(1+(beta*t_actual))*

●Búsqueda Tabú.

Inicializo los parámetros:

tamtabu <- n/2

listatabu <- matriz n_x4 (n unidades : sol[r], sol[s], r, s)

listaveci <- matriz 30x5 (30vecinos : sol[r], sol[s], r, s, difcoste cambio r,s)

cont <- 0

Bucle principal:

Mientras cont < 25000

Genero 30 vecinos aleatorios ordenados según la diferencia de coste.

Me quedo con el mejor vecino no tabú o que supere el criterio de aspiración (costedif negativo)

Si el mejor vecino no es tabú:

solución actual será la solución vecina

obtengo coste de solución actual

Si coste de solución actual es mejor que coste solución global

solución global <- solución actual

añado cambio de solución vecina al final de lista tabú

Si el mejor vecino es tabú:

Si supera nivel de aspiración (costedif negativo)

solución actual será la solución vecina

obtengo coste de solución actual

Si coste de solución actual < coste solución global

solución global <- solución actual

añado cambio de mejor vecino al final de lista tabú

Calculo el coste de la solución.

●Búsqueda Tabú Extendida.

Inicializo los parámetros:

tamtabu <- n/2, cont <- 0

listatabu <- matriz n_x4 (n unidades : sol[r], sol[s], r, s)

listaveci <- matriz 30_x5 (30vecinos : sol[r], sol[s], r, s, difcoste cambio r,s)

mem_largo_plazo <- matriz n_xn simétrica

Bucle principal:

Mientras cont < 25000

Genero 30 vecinos aleatorios ordenados según la diferencia de coste.

Me quedo con el mejor vecino no tabú o que supere el criterio de aspiración (costedif negativo)

Si el mejor vecino no es tabú:

solución actual será la solución vecina

obtengo coste de solución actual

Si coste de solución actual es mejor que coste solución global

solución global <- solución actual

añado cambio de solución vecina al final de lista tabú

Si el mejor vecino es tabú:

Si supera nivel de aspiración (costedif negativo)

solución actual será la solución vecina

obtengo coste de solución actual

Si coste de solución actual < coste solución global

solución global <- solución actual

añado cambio mejor vecino al final de lista tabú

Actualizo la memoria a largo plazo

Si llevo 10 iteraciones sin mejorar, hago reinicialización aleatoria

Obtengo la mejor solución global

Genero aleatorio entre [1,4]

25% → Solución actual será solución aleatoria

25% → Solución actual será mejor solución global

50% → Solución usando los valores menos visitados de memoria de frecuencias.

Calculo el coste de la solución.

4. Experimentos y análisis de resultados

Algoritmo Greedy			
Caso	Coste obtenido	Desv	Tiempo
Chr20a	8100	269,53	0,00
Chr20c	78718	456,63	0,00
Chr22b	11942	92,80	0,00
Chr25a	17556	362,49	0,00
Esc32a	342	163,08	0,00
Esc64a	148	27,59	0,00
Esc128	142	121,88	0,00
Kra32	117130	32,05	0,00
Lipa90a	367797	1,99	0,00
Sko42	18902	19,54	0,00
Sko49	27378	17,07	0,00
Sko81	105828	16,30	0,00
Sko90	131450	13,78	0,00
Sko100f	170472	21,17	0,00
Tai64c	5893540	217,55	0,00
Tai80a	15638632	15,85	0,00
Tai100a	23926646	51,01	0,00
Tai150b	621314634	40,64	0,00
Tai256c	98685678	125,05	0,00
Tho150	9527466	25,02	0,00

Se puede ver que las soluciones obtenidas mediante Greedy, como era de esperar, son muy malas. Parece ser que los problemas con menos tamaño obtienen peores resultados que aquellos que tienen mayor tamaño. Sin embargo, los tiempos son tan pequeños que no llega ni a suponer milisegundos por problema.

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

Mediante búsqueda local vemos que hay algunos casos en los que obtenemos resultados bastante buenos (desde kra32 hasta tai80a aprox). Sin embargo, para problemas con muchas o pocas unidades a asociar, los resultados no son aceptables. Los tiempos siguen siendo muy buenos, sin llegar ni siquiera a tardar un segundo en el mayor de los problemas.

Algoritmo ES			
Caso	Coste obtenido	Desv	Tiempo
Chr20a	3306	50,82	0.006274
Chr20c	23280	64,62	0.002331
Chr22b	6924	11,79	0.002333
Chr25a	5256	38,46	0.004242
Esc32a	148	13,85	0.100498
Esc64a	116	0,00	0.148715
Esc128	94	46,88	0.471415
Kra32	94340	6,36	0.015898
Lipa90a	363550	0,81	0.622043
Sko42	16362	3,48	0.059112
Sko49	23866	2,05	0.082247
Sko81	92744	1,92	0.437560
Sko90	118110	2,23	0.601359
Sko100f	151554	7,72	0.693596
Tai64c	1876252	1,10	0.142611
Tai80a	14071292	4,24	0.437677
Tai100a	21990420	38,79	0.289514
Tai150b	517634873	17,17	2.388852
Tai256c	45706204	4,23	2.342876
Tho150	8370098	9,83	2.357514

Usando dicho algoritmo, vemos que hay casos en los que los resultados son bastante buenos, incluso en el caso esc64a conseguimos alcazar la solución óptima. Los peores resultados los encontramos al trabajar con casos en los que hay pocas unidades a asociar.
Los tiempos siguen siendo muy buenos: sólamente nos encontramos tres casos en los que el tiempo de ejecución supera unos segundos.

Algoritmo BT			
Caso	Coste obtenido	Desv	Tiempo
Chr20a	3450	57,39	0.562586
Chr20c	20808	47,14	0.556570
Chr22b	6736	8,75	0.628098
Chr25a	5350	40,94	0.686665
Esc32a	152	16,92	0.847517
Esc64a	116	0,00	1.785796
Esc128	72	12,50	4.525726
Kra32	97260	9,65	0.847421
Lipa90a	363493	0,79	2.791038
Sko42	16372	3,54	1.109585
Sko49	23910	2,24	1.325514
Sko81	93484	2,73	2.461597
Sko90	118310	2,40	2.793331
Sko100f	151330	7,56	3.215280
Tai64c	1866152	0,55	1.776840
Tai80a	14075542	4,27	2.393140
Tai100a	21668034	36,75	3.208422
Tai150b	514202556	16,39	5.820798
Tai256c	44958438	2,53	13.879650
Tho150	8350126	9,57	5.695707

Usando el algoritmo de búsqueda tabú básico (con memoria a corto plazo), obtenemos unos resultado en general muy buenos. Volvemos a encontrarnos varios casos en los que se alcanza el óptimo o solución muy cercanas a la óptima. No obstante, en problemas con pocas unidades tenemos resultados malos, llegando incluso a una desviación de 57.39 en el primer caso.
En cuanto a los tiempo vemos que conforme el tamaño del problema aumenta, aumentan significativamente. Sin embargo, esto no supone ningún problema pues como máximo se tarda 13.8 segundos.

Algoritmo BTExt			
Caso	Coste obtenido	Desv	Tiempo
Chr20a	2848	29,93	0.570910
Chr20c	14142	0,00	0.569848
Chr22b	7526	21,50	0.628098
Chr25a	6058	59,59	0.692062
Esc32a	148	13,85	0.888487
Esc64a	116	0,00	2.088114
Esc128	76	18,75	6.473264
Kra32	95430	7,59	0.887915
Lipa90a	363886	0,90	3.402645
Sko42	16100	1,82	1.199940
Sko49	23948	2,40	1.414975
Sko81	92772	1,95	2.729606
Sko90	120146	3,99	3.237242
Sko100f	155502	10,53	3.747586
Tai64c	2077120	11,92	2.147522
Tai80a	14312890	6,03	2.771511
Tai100a	21771040	37,40	3.885452
Tai150b	548343023	24,12	6.956269
Tai256c	46180354	5,32	27.195871
Tho150	8698446	14,14	7.019523

Mediante el algoritmo tabú extendido (haciendo uso de memoria a corto y largo plazo) vemos que tenemos muy buenos resultados. Apenas hay casos en los que dispare la desviación negativamente y hay bastantes casos en los que se obtiene una solución cercana a la óptima.
En cuanto a los tiempos, vemos siguen siendo buenos pero se incrementa notablemente en el caso con 256 unidades. Esto es comprensible pues con dicho algoritmo estamos realizando muchas comprobaciones y el tamaño del problema es grande. Aún así, merece la pena pues la desviación que obtenemos es muy pequeña.

Veamos la comparación entre los algoritmos:

Algoritmo	Desv	Tiempo
Greedy	104,55	0
BL	17,09	0
ES	16,32	0
BT básica	14,13	0
BT extendida	13,59	0

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 trabajar con el algoritmo de búsqueda local, vemos que la desviación en media es considerablemente buena. Enfriamiento simulado obtiene resultados mejores que búsqueda local, pues permite salir de los óptimos locales en los que la búsqueda local se puede ver frenada. Aunque se permita ir hacia peores soluciones, obtenemos soluciones mejores que en búsqueda local pues a medida que vamos avanzando en el algoritmo vamos evitando ir hacia soluciones peores. De este modo, no vamos a obtener resultados gravemente malos por el echo de ir a una solución peor al final del proceso. No obstante, se podría dar el caso en el que ES obtuviese peores resultados en media que BL.

A continuación, vemos que la búsqueda tabú básica mejora todos los algoritmos anteriores en media. Esto se debe a que, a diferencia de lo que hacíamos en ES, al generar vecinos evitamos la exploración de zonas ya visitadas. Es decir, generamos entornos tabú restringidos.

Aún así, es claro que también podemos ir hacia soluciones peores, por lo que dicho algoritmo no nos garantiza que siempre vaya a tener soluciones mejores que los anteriores.

Finalmente, vemos los resultados de búsqueda tabú extendida. Hemos tenido suerte, y ha obtenido mejores resultados que en cualquiera de los otros algoritmos. Esto no siempre ha de ser así. En este caso, las reinicializaciones de soluciones nos han servido para salir de óptimos locales y llevar a óptimos locales mejores o incluso a la solución óptima.

Valorando casos particulares:

Fijándonos en los resultados obtenidos, vemos que el caso chr25a es bastante problemático en el sentido de que con cualquiera de los algoritmos obtenemos soluciones muy malas. Tendremos que realizar otra serie de algoritmos para conseguir aproximarnos más a la solución óptima. Es un caso que nos está produciendo un gran aumento de desviación en todos los algoritmos que hemos visto.

Por otro lado, nos encontramos el caso lipa90a que nos da resultados muy buenos para cualquiera de los algoritmos, incluso para Greedy.

Número de veces que es mejor…:

Aunque los algoritmos BL, ES, BT, BText vayan mejorando los tiempos, 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
- BL: 2
- ES: 6
- BT: 7
- BTExt: 5

Parece curioso el hecho de que la búsqueda tabú extendida tenga un menor número veces en el que es mejor con respecto a ES y la búsqueda tabú básica. Recordemos que BText obtenía la desviación más pequeña en media.
Esto se debe a que, aunque 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:
  BL: 14.27
  ES: 18.23
  BT: 15.17
  BText: 13.48

- Con semilla 1111:
  BL: 16.31
  ES: 16.88
  BT: 17.08
  BText: 12.78

- Con semilla 9090:
  BL: 16.53
  ES: 15.48
  BT: 17.82
  BText: 16.21

Estos resultados nos permiten ver que un algoritmo un tiene porqué ser mejor que otro. No podemos asociar un algoritmo a un problema. La elección del algoritmo dependerá del caso concreto del problema. Tal vez un algoritmo sea mejor para un caso, pero peor para otro.

5. Procedimiento para el desarrollo de la práctica

Para la realización de la práctica he hecho uso de los pdfs seminario2a, 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.
Además he usado los códigos de ayuda que se encuentran disponibles en la plataforma.