Práctica 5.a: Búsquedas Híbridas para el Problema de la Asignación Cuadrática

Alejandro Alcalde*

Tercer curso Grado en Ingeniería Informática, Granada, grupo de los miércoles a las 17.30

algui91@gmail.com

Algoritmos: MA(10,1), MA(10, .1), MA(10, .1best)

Índice

*Template by howtoTeX.com, elbauldelprogramador.com

Resumen

Memoria de la quinta práctica de la asignatura MetaHeurísticas del tercer curso del Grado en Ingeniería Informatica de la facultad de Granada. La practica consiste en la implementación de tres versiones de algoritmos meméticos usando el genético generacional de la práctica tercera junto a la búsqueda local de la práctica primera.

I. Descripción del problema

El problema de la asignación cuadrática (QAP) es un problema estándar en teoría de localización. En éste se trata de asignar N unidades a una cantidad N de sitios o localizaciones en donde se considera un costo asociado a cada una de las asignaciones. Este costo dependerá de las distancias y flujo entre las unidades, además de un costo adicional por asignar cierta unidad a cierta localización específica. De este modo se buscará que este costo, en función de la distancia y flujo, sea mínimo.

I.1. Definición matemática

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

Donde $\prod_N$ es el conjunto de todas las permutaciones posibles de $N = 1, 2, \ldots, n$

I.2. Representación del problema

En éste problema las soluciones se pueden representar como permutaciones de un conjunto. Si el problema es de tamaño cuatro, por ejemplo, una solución vendría dada por la permutación $N = \{3, 2, 1, 4\}$. Si tomamos los índices de éste conjunto como las unidades, y el valor en dicho índice como localizaciones, la localización 3 estaría asignada a la unidad 1, la localización 2 a la unidad 2 etc.

El objetivo del problema es **minimizar** la expresión mostrada anteriormente. Ésta será la función objetivo:

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

Hará falta un mecanismo para **generar la solución inicial**, en nuestro caso, la permutación N inicial sobre la que lanzar los distintos algoritmos será aleatoria.

De igual modo será necesario un esquema de **generación de soluciones vecinas**. Para ello, se realizará un intercambio entre dos localizaciones, cambiando sus respectivas unidades por la otra.

II. Composición de los algoritmos

II.1. Esquema de representación

Como se ha mencionado anteriormente, el esquema elegido para representar una solución ha sido una permutación del tipo $\{1, 2, 3, 4\}$ en la que los elementos representan localizaciones y la posición que ocupan las unidades a las que han sido asignadas.

II.2. Descripción de la función objetivo

La función objetivo descrita con su fórmula en apartados anteriores puede representarse con el siguiente pseudocódigo:

```
i = 0 .. Tamaño del problema
    j = 0 .. Tamaño del problema
        acumular el coste producido al asociar el flujo existente entre
        la unidad i y j y la distancia existente entre las localizaciones
        i y j
```

II.3. Generación de soluciones aleatorias

El tamaño de la población para esta práctica debe ser de 10, por lo tanto es necesario generar 10 soluciones iniciales aleatorias. Para así disponer de una primera población. El mecanismo seguido para obtener la población inicial, dada una semilla inicial, es el siguiente:

```
desde i=0 hasta el tamaño de la población
    solucion aleatoria = []
    mientras el tamaño de la solución no sea correcto
        generar un número aleatorio entre 0 y el tamaño de la solución
            mientras el número generado ya esté en la solución
                descartarlo y generar uno nuevo
    añadir la solución aleatoria a la población
```

Éste proceso se realiza al principio de la ejecución del algoritmo.

II.4. Algoritmo de búsqueda local empleado

```
mientras el numero de evaluaciones < (el máx de eval Y improve_flag)
    poner improve_flag a falso
    iterar mientras i < tam problema y no improve_flag
        si en la tabla DLB el elemento i está a 0
            iterar mientras j < tam. Problema y no improve_flag
                delta = factorización(i,j)
                incrementar numero de evaluaciones
                si hubo mejora # delta < 0
                    actualizar el coste actual
                    aplicar el intercambio i,j
                    establecer la DLB para i,j a 0.
                    fijar improve_flag a verdadero
            si no se mejoró # improve_flag a falso
                establecer la DLB para i a 1.
```

II.4.1. Exploración del entorno

Se emplea una técnica llamada *Don't Look Bits*, que permite focalizar la Búsqueda en una zona del espacio en la que se puede encontrar una solución mejor que la actual.

Con ésta máscara marcamos con 0 todas la unidades inicialmente, indicando que se exploran todos en orden secuencial. El 1 indica que dicho vecino no va a ser explorado.

```
        DLB = 1..N inicializado a 0
        for i = 1 .. n
            si DLB_i == 0 # se puede mirar el vecino
                for j = 1 .. n
```

```
                vecino = intercambio(i,j)
                comparar(vecino, solucion actual)
                Si mejora
                    solucion actual = vecino
                    DLB_i = DLB_j = 0
```

II.4.2. Operador de generación de vecinos

El operador de vecino simplemente intercambia una posición de la solución con otra:

```
s = solución
i,j los dos índices a intercambiar
s_i , s_j = s_j , s_i # Se intercambian los valores s_i , s_j = s_j , s_i
```

II.4.3. Factorización

Tras aplicar el operador de vecino, en lugar de calcular de nuevo el coste mediante la función objetivo, se realiza una factorización que reduce el tiempo de cálculo considerablemente, ya que evalua únicamente el incremento o decremento en el coste al aplicar un intercambio entre dos elementos, r y s.

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

III. Estructura del método de búsqueda

III.1. MA(10,1)

El método de búsqueda para esta versión consiste en ejecutar el algoritmo genético generacional con una población de 10 cromosomas. Cada 10 generaciones se aplicará la búsqueda local sobre todos los individuos con un máximo de 400 evaluaciones para cada uno. El pseudocódigo es el siguiente:

```
mientras no se cumpla el criterio de parada
    intercambiar la población antigua con la nueva
    seleccionar individuos a formar parte de la nueva generación
    cruzar algunos individuos
    mutar algunos individuos
    reemplazar la población siguiendo esquema elitista.
    evaluar la población nueva

    cada 10 generaciones
        para cada elemento de la población
            aplicar busqueda local con máximo de 400 evaluaciones
            restar las evaluaciones de la BL al criterio de parada
            contador de generaciones a 0
```

III.2. MA(10, 0.1)

Ésta versión, al igual que la anterior, ejecuta el genético generacional, cada 10 generaciones aplica la búsqueda local con 400 evaluaciones, pero ahora se aplica sobre el 10% de la población. Es decir, $0{,}1 \cdot 10$ (Se ha hecho así para ahorrar la generación de números aleatorios adicionales)

```
mientras no se cumpla el criterio de parada
    intercambiar la población antigua con la nueva
    seleccionar individuos a formar parte de la nueva generación
    cruzar algunos individuos
    mutar algunos individuos
    reemplazar la población siguiendo esquema elitista.
    evaluar la población nueva

    cada 10 generaciones
        para i = 0 hasta 0.1 * 10
            aplicar BL al elemento i de la población
            restar las evaluaciones de la BL al criterio de parada
            contador de generaciones a 0
```

III.3. MA(10, 0.1Mejor)

Ésta versión, al igual las anteriores, ejecuta el genético generacional, cada 10 generaciones aplica la búsqueda local con 400 evaluaciones, pero ahora se aplica sobre el 10 % de los mejores individuos.

```
mientras no se cumpla el criterio de parada
    intercambiar la población antigua con la nueva
    seleccionar individuos a formar parte de la nueva generación
    cruzar algunos individuos
    mutar algunos individuos
    reemplazar la población siguiendo esquema elitista.
    evaluar la población nueva

    cada 10 generaciones
        obtener los 0.1 * 10 mejores individuos
        guardar los índices de dichos individuos
        para i = cada índice de los mejores individuos
            aplicar BL al elemento i de la población
            restar las evaluaciones de la BL al criterio de parada
            contador de generaciones a 0
```

IV. Algoritmo de comparación, Greedy

Consiste simplemente en asociar unidades de gran flujo con localizaciones céntricas en la red y viceversa. Para ello se calcula el potencial de flujo y de distancia. A mayor potencial de flujo, más peso tendrá dicha unidad en el intercambio de flujos y, a menor flujo de distancia, más céntrica será la localización. Por tanto el algoritmo selecciona la unidad disponible con mayor potencial de flujo y le asignará la localización de menor potencial de distancia.

```
    Calculo potenciales.
    Inicialzar N a 0.
    para i = 1 hasta el tamaño del problema
        Seleccionar unidad de mayor potencial de flujo
        Seleccionar localización de menor potencial de distancia
        Añadir la localización seleccionada al índice correspondiente a la unidad en N
```

V. Procedimiento considerado

La implementación se ha realizado en Python, basándose en las explicaciones de clase y los documentos proporcionados por el profesorado. Para ejecutar el programa:

```
$ python QAP.py -d <datos del problema> -a \
[ma_10_1_pmx | ma_10_01_pmx | ma_10_01best_pmx] -s semilla -v<verbose>
```

VI. Experimentos y análisis de resultados

Para esta pactica se han usado todos los casos, salvo los bur. La semilla usada fue 3327229832
A continuación se muestra la tabla con los resultados para los distintos algoritmos.

Algoritmo	Desviación	Tiempo
Greedy	81.4492028061	0.0012273788
MA(10,1)	10.8358796412	1.5203776227
MA(10,0.1)	9.4251761779	0.0043138133
MA(10,0.1mej)	10.6540082225	0.0047982401

Cuadro 1

MA(10, 1))			
Caso	Coste	Desv	Tiempo
Els19	17997928	4.56	0.7779898643
Chr20a	2706	23.45	0.7962989807
Chr25a	5208	37.20	0.6448259354
Nug25	3798	1.44	0.935076952
Tai30a	1915336	5.35	0.7464430332
Tai30b	768457665	20.61	1.3706448078
Esc32a	154	18.46	0.8313019276
Kra32	94690	6.75	0.9608359337
Tai35a	2552422	5.38	1.0090508461
Tai35b	327182957	15.48	1.2285130024
Tho40	263814	9.69	1.5948040485
Tai40a	3333080	6.17	1.4561870098
Sko42	16820	6.37	1.6261639595
Sko49	24924	6.58	1.8619780541
Tai50a	5287240	7.06	1.5881719589
Tai50b	511149312	11.40	2.2749328613
Tai60a	7779014	7.95	2.8702600002
Lipa90a	364706	1.13	4.7933180332

Cuadro 2

VI.1. Observaciones

El mejor algoritmo globalmente es `MA(10,0.1)` probablemente porque tenga un mayor equilibrio entre explotación y exploración. Ya que solo realizará una explotación mediante búsqueda local a un

MA(10, 0.1)			
Caso	Coste	Desv	Tiempo
Els19	18124442	5.30	0.001611948
Chr20a	2714	23.81	0.0017259121
Chr25a	5664	49.21	0.0017328262
Nug25	3788	1.18	0.0025629997
Tai30a	1888706	3.88	0.002959013
Tai30b	769103949	20.72	0.0029649734
Esc32a	150	15.38	0.0038039684
Kra32	94920	7.01	0.0034320354
Tai35a	2526558	4.32	0.0039310455
Tai35b	303551155	7.14	0.0040099621
Tho40	246858	2.64	0.0048840046
Tai40a	3265130	4.01	0.0042901039
Sko42	16500	4.35	0.0051498413
Sko49	24164	3.33	0.0058801174
Tai50a	5113536	3.54	0.0062999725
Tai50b	489069535	6.59	0.005863905
Tai60a	7659162	6.29	0.0055539608
Lipa90a	364101	0.96	0.0109920502

Cuadro 3

10 % de la población, lo cual deja bastante margen al algoritmo genético para explorar el espacio de soluciones y no quedar estancado localmente.

En algunas instancias concretas, `MA(10,0.1mej)` y `MA(10,0.1)`, obtienen resultados muy cercanos al óptimo. Ejemplos de instancias en las que ocurre esto, para `MA(10,0.1)` son **Nug25** y **Lipa90a**, con desviaciones de 1.18 y 0.96 respectivamente. Esto se puede deber a que el algoritmo genético haya realizado una buena exploración y se haya situado en la región del óptimo, para despues haber explotado ese espacio con la búsqueda local. Para `MA(10,0.1mej)` vuelve a destacar la cercanía con el óptimo de estas dos instancias, en este caso 0.64 para **Nug25** y 1.01 para **Lipa90a**

MA(10, 0.1mej)			
Caso	Coste	Desv	Tiempo
Els19	18039214	4.80	0.0016949177
Chr20a	3084	40.69	0.0018479824
Chr25a	6366	67.70	0.0027389526
Nug25	3768	0.64	0.0025930405
Tai30a	1887820	3.83	0.002820015
Tai30b	697468377	9.47	0.0029940605
Esc32a	150	15.38	0.0032198429
Kra32	92090	3.82	0.0036840439
Tai35a	2530044	4.46	0.0039908886
Tai35b	305858719	7.96	0.0041918755
Tho40	251344	4.50	0.0054271221
Tai40a	3329006	6.04	0.0041720867
Sko42	16432	3.92	0.0033957958
Sko49	24150	3.27	0.0056490898
Tai50a	5180742	4.90	0.0076389313
Tai50b	478146849	4.21	0.0064709187
Tai60a	7577504	5.16	0.0082199574
Lipa90a	364257	1.01	0.0156188011

Cuadro 4