Práctica 1.b: Técnicas de Búsqueda Local y Algoritmos Greedy para el Problema del Agrupamiento con Restricciones:

UNIVERSIDAD DE GRANADA

Luis González Romero, XXXXXXXXX, luisgonromero@correo.ugr.es
Grupo 1: Miércoles 17:30-19:30

Escuela Técnica Superior de Ingeniería informática y Telecomunicaciones

23 de marzo de 2020

Práctica 1.b: Técnicas de Búsqueda Local y Algoritmos Greedy para el Problema del Agrupamiento con Restricciones

Memoria sobre la primera práctica de la asignatura Metaheurísticas cursada en la ETSIIT, de la UGR.

Luis González Romero, XXXXXXXXX, luisgonromero@correo.ugr.es

Marzo de 2020

Índice

Índice de figuras

Índice de tablas

1. Descripción del problema: Problema del Agrupamiento con Restricciones

El problema del *PAR* consiste en una generalización del agrupamiento clásico. Permite incorporar al proceso de agrupamiento un nuevo tipo de información: las restricciones. Dado un conjunto de datos X con n instancias, el problema consiste en encontrar una partición $C = \{c_1, ..., c_{1k}\}$ que minimice la desviación general y cumpla con las restricciones de instancia existentes en el conjunto R. Dada una pareja de instancias, se establece una restricción de tipo *Must-Link* (ML) si deben pertenecer al mismo grupo y de tipo *Cannot-Link* (CL) si no pueden pertenecer al mismo grupo.

Dada una partición C, se puede calcular el centroide $\overrightarrow{\mu}_i$ asociado a cada grupo c_i como el vector promedio de las instancias de X que lo componen:

$$\overrightarrow{\mu}_i = \frac{1}{|c_i|} \sum_{\overrightarrow{x}_j \in c_i} \overrightarrow{x}_j$$

La distancia media intra-cluster $\overrightarrow{ci}_i$ será la media de las distancias de las instancias que lo conforman a su centroide:

$$\overline{c}_i = \frac{1}{|c_i|} \sum_{\overrightarrow{x}_j \in c_i} \| \overrightarrow{x}_j - \overrightarrow{\mu}_i \|_2$$

Definimos la desviación general de la partición $C = \{c_1, ..., c_{1k}\}$ como la media de las desviaciones intracluster $\overline{C}$:

$$\overline{C} = \frac{1}{k} \sum_{c_i \in C} \overline{c_i}$$

Dada una partición C y los conjuntos de restricciones ML y CL, definimos infeasibility ("infactibilidad") como el número de restricciones que C incumple.

Así, se puede formular como:

$$Minimizar f = \overrightarrow{C} + (infeasability * \lambda)$$

siendo λ un parámetro de escalado para dar relevancia a la infeasibility. Así poder optimizar simultáneamente el número de restricciones incumplidas y la desviación general. Para asegurar que el factor infeasibility tiene la suficiente relevancia establecemos λ como el cociente entre la distancia máxima existente en el conjunto de datos y el número de restricciones presentes en el problema, $|R|$.

2. Algoritmos empleados al problema, consideraciones comunes

Estructuras de datos comunes: *data* $\rightarrow$ se almacenan los datos de las instancias del dataset *restricciones* $\rightarrow$ se almacenan las restricciones en forma de matriz *restricciones_lista* $\rightarrow$ se almacenan las restricciones en forma de lista $RSI \rightarrow$ array de tamaño n, el cual es barajado para poder generar diversidad en las iteraciones *k_distancias* $\rightarrow$ se almacenan las distancias de cada instancia a cada uno de los centroides.

No me da tiempo a escribir el pseudocódigo de los métodos comunes. Pero serían el calculo de la infeasibility recorriendo la matriz por encima de la diagonal e ir incrementando si se encuentra un *ML* o *CL*. Junto a los métodos de actualización de estructuras de datos.

3. Algoritmo de Búsqueda Local

Estructuras de datos utilizadas:
$S \rightarrow$ lista de tamaño n_instancias, para cada i se almacena el cluster más cercano
vecinos $\rightarrow$ array que contiene el entorno virtual generado por los vecinos de S, de la forma $pair_i(instancia, nuevo\_cluster)$

procedure BUSQUEDA_LOCAL
generar_solucion_inicial_aleatoria()
generar_vecinos()
$acceso\_aleatorio \leftarrow lista(0, ..., n\_vecinos)$
barajar(acceso_aleatorio)
$f\_actual \leftarrow f()$
while $end \neq TRUE$ AND $evaluations < 100000$ **do**
$end \leftarrow TRUE$
for $i = 0$ to *n_vecinos-1* **do**
$instancia\_actual \leftarrow vecinos[acceso\_aleatorio[i]].get < 0 >$
$nuevo\_cluster \leftarrow vecinos[acceso\_aleatorio[i]].get < 1 >$
$cluster\_instancia\_actual \leftarrow S[instancia_actual]$
$S[instancia\_actual] \leftarrow nuevo\_cluster$
if validar_solucion_actual() == TRUE **then**
$f\_vecino \leftarrow f\_actual$
$evaluaciones \leftarrow evaluaciones + 1$
if $f\_vecino < f\_actual$ **then**
actualizar_centroides()
$f\_actual \leftarrow f\_vecino$
generar_vecinos()
barajar(acceso_aleatorio)
$end \leftarrow FALSE$
break
else
$S[instancia\_actual] \leftarrow nuevo\_cluster$
end if
else
$S[instancia\_actual] \leftarrow cluster\_instancia\_actual$
end if
end for
end while
end procedure

Para generar la primera solución he creado un método el cual rellena la lista S de

clusters aleatorios de *0* a *k-1*. Después comprueba si la solución es válida(si ningún cluster está vacío), si lo es devuelve *True* y si no *False*. Si no es válida, se generan soluciones aleatorias hasta que una lo sea.

```
procedure GENERAR_SOLUCION_INICIAL_ALEATORIA()
    for i = 0 to n_instancias-1 do
        S.append(aleatorio(0, k-1)
        if validar_solucion_actual() == False then
            S.clear()
            for i = 0 to n_instancias-1 do
                S.append(aleatorio(0, k-1)
            end for
        end if
    end for
end procedure
```

Para la solución al problema mediante BL, he seguido la estrategia *el primer mejor*. Exploro el entorno virtual de forma aleatoria gracias a *acceso_aleatorio* quedándome con una nueva solución S, en cuanto el primer vecino mejore la *f()* actual. El proceso finaliza en cuanto se explore un entorno virtual entero sin mejora o se lleguen a las 100000 evaluaciones, lo que antes suceda.

Para lograr que haya diversidad, acceso_aleatorio es barajado cada vez que se genera un vecindario virtual.

El operador de vecino se realiza antes de evaluar al vecino por acceso aleatorio, guardando una copia del cluster que pertenecía a *S[vecino_aleatorio]*. Así, si no hay mejora se puede devolver S a su estado anterior.

Además, en el método de infeasibility usado en BL, se usa la estructura de datos que guarda las restricciones en una lista(mayor eficiencia). Evitando así todas las iteraciones por encima o debajo de la diagonal de la matriz(como se hace en el greedy). Y en cuanto a la actualización de centroides, en el BL hay una versión que solo actualiza el cluster nuevo que se ha cambiado en *S[i]* y el anterior que habia en *S[i]*.

Y en cuanto a la generación de vecinos, creo un vecindario virtual(pares de valores, instancia-nuevo_cluster) como ya he comentado, guardando solo vecinos válidos

4. Algoritmo COPKMN - Greedy

Estructuras de datos utilizadas:

$RSI \rightarrow$ indices aleatorios

$local_iv \rightarrow$ infeasibility local para un cluster e instancias dadas

$min\_ci \rightarrow$ cada posicion i tiene el cluster de local_iv[i]

$distancias \rightarrow$ cada posicion i tiene la distancia al cluster cluster_minimo

$indices\_minimo \rightarrow$ indice(s) de valor minimo

$c\_array \rightarrow c\_array[k][n]$ que contiene k arrays para almacenar las instancias de datos que pertenecen a cada cluster

$cluster\_mas\_cercano \rightarrow$ tiene tamaño n_instancias en el que cada i se almacena el cluster más cercano para la instancia i

procedure COPKMN
    crear_cluster_cercanos()
    $cambio \leftarrow True$
    $i \leftarrow 1$
    while $cambio == TRUE$ **do**
        $cambio \leftarrow 0$
        $i \leftarrow i + 1$
        if *cambia_cluster_mas_cercano()* $== TRUE$ **then**
            $cambio \leftarrow 1$
        end if
        actualizar_centroides()
    end while
end procedure

function *cambia_cluster_mas_cercano()*
    actualizar_distancias()
    return *actualizar_cluster_mas_cercano()*
end function

Método para obtener el cluster más cercano:

function *cambia_cluster_mas_cercano()*
    $cambio \leftarrow False$
    for $i = 0$ to *n_instancias* **do**
        $acceso\_aleatorio \leftarrow RSI[i]$
        $distancia\_minima \leftarrow 999$
        $min\_i \leftarrow cluster\_mas\_cercano[acceso\_aleatorio]$
        for $j = 0$ to *k-1* **do**
            $inf\_valor \leftarrow infeasibility(j, random_access)$

```
            local_iv.append(iv)
            min_ci.append(j)
            distancias.append(k_distancias[acceso_aleatorio][j])
        end for
        minimo ← min(local_iv)
        for indice, elemento in enumerate(local_iv) do
            if minimo == elemento then
                indices_minimo.append(indice)
            end if
        end for
        for a = 0 to len(indices_minimo do
            minimo_temporal ← distancias[indices_minimo[a]]
            if minimo_temporal < distancia_minima then
                distancia_minima ← minimo_temporal
                instancia_minima ← indices_minimo[a]
            end if
        end for
        if cluster_mas_cercano[acceso_aleatorio] ≠ instancia_minima then
            if cluster_mas_cercano[acceso_aleatorio] ≠ 999 then
                c_array[cluster_mas_cercano[acceso_aleatorio]].remove(acceso_aleatorio)
            end if
            c_array[instancia_minima].append(acceso_aleatorio)
            cluster_mas_cercano[acceso_aleatorio] ← cluster_minimo[instancia_minima]
            cambio ← 1
        end if
    end for
    return cambio
end function
```

El algoritmo greedy es muy simple. Se busca un centroide más cercano, iterando de forma aleatoria sobre todo el dataset. Se cambia la solución actual por una nueva y se vuelven a actualizar los centroides y las distancias a estos.

5. Análisis de resultados

Tablas obtenidas

	Iris				Ecoli				Rand			
	Tasa_C	Tasa_Inf	Agr.	T	Tasa_C	Tasa_Inf	Agr.	T	Tasa_C	Tasa_Inf	Agr.	T
Ejecución 1	0.67	0	x	0.06	38.33	480	x	0.61	0.76	0	x	0.05
Ejecución 2	0.67	0	x	0.08	37.45	861	x	1.5	0.76	0	x	0.06
Ejecución 3	0.67	0	x	0.07	39.84	1124	x	1	0.76	0	x	0.05
Ejecución 4	0.67	0	x	0.07	38.22	674	x	0.66	0.76	0	x	0.04
Ejecución 5	0.67	0	x	0.10	39.25	477	x	0.6	0.76	0	x	0.04
Media	0.67	0	x	0.07	38.62	723.2	x	0.87	0.76	0	x	0.05

Tabla 1: Resultados obtenidos por el algoritmo COPKMN en el PAR con 10 % de restricciones

	Iris				Ecoli				Rand			
	Tasa_C	Tasa_Inf	Agr.	T	Tasa_C	Tasa_Inf	Agr.	T	Tasa_C	Tasa_Inf	Agr.	T
Ejecución 1	0.67	0	x	0.03	36.35	669	x	0.98	0.76	0	x	0.03
Ejecución 2	0.67	0	x	0.04	35.88	360	x	1.57	0.76	0	x	0.03
Ejecución 3	0.67	0	x	0.03	37.51	365	x	0.93	0.76	0	x	0.03
Ejecución 4	0.67	0	x	0.03	35.98	632	x	0.8	0.76	0	x	0.03
Ejecución 5	0.67	0	x	0.4	36.85	1138	x	1.5	0.76	0	x	0.03
Media	0.67	0	x	0.03	36.51	632.8	x	1.16	0.76	0	x	0.03

Tabla 2: Resultados obtenidos por el algoritmo COPKMN en el PAR con 20 % de restricciones

	Iris				Ecoli				Rand			
	Tasa_C	Tasa_Inf	Agr.	T	Tasa_C	Tasa_Inf	Agr.	T	Tasa_C	Tasa_Inf	Agr.	T
Ejecución 1	0.67	0	0.67	2.10	23.21	64	24.93	96.66	0.76	0	0.76	1.42
Ejecución 2	0.67	0	0.67	1.71	26.34	86	28.64	55.65	0.76	0	0.76	2.17
Ejecución 3	0.67	0	0.67	1.94	23.91	58	25.46	102.65	0.76	0	0.76	1.31
Ejecución 4	0.67	0	0.67	1.70	23.69	74	25.68	123.89	0.76	0	0.76	1.69
Ejecución 5	0.67	0	0.67	2.04	23.11	74	25.10	93.03	0.76	0	0.76	1.50
Media	0.67	0	0.67	1.90	24.05	71.2	25.96	94.38	0.76	0	0.76	1.62

Tabla 3: Resultados obtenidos por el algoritmo BL en el PAR con 10 % de restricciones

	Iris				Ecoli				Rand			
	Tasa_C	Tasa_Inf	Agr.	T	Tasa_C	Tasa_Inf	Agr.	T	Tasa_C	Tasa_Inf	Agr.	T
Ejecución 1	0.67	0	0.67	3.09	22.94	146	24.89	168.72	0.76	0	0.76	2.33
Ejecución 2	0.67	0	0.67	2.97	23.07	170	25.35	137.05	0.76	0	0.76	1.90
Ejecución 3	0.67	0	0.67	2.45	23.02	218	25.94	148.29	0.76	0	0.76	2.46
Ejecución 4	0.67	0	0.67	2.86	22.90	208	25.69	124.18	0.76	0	0.76	2.82
Ejecución 5	0.67	0	0.67	2.89	23.02	204	25.76	187.39	0.76	0	0.76	2.58
Media	0.67	0	0.67	2.85	22.99	189.2	25.53	153.13	0.76	0	0.76	2.42

Tabla 4: Resultados obtenidos por el algoritmo BL en el PAR con 20 % de restricciones

	Iris				Ecoli				Rand			
	Tasa_C	Tasa_Inf	Agr.	T	Tasa_C	Tasa_Inf	Agr.	T	Tasa_C	Tasa_Inf	Agr.	T
COKPMN	0.67	0	x	0.06	38.33	723.20	x	0.87	0.76	0	x	0.05
BL	0.67	0	0.67	1.90	24.05	71.20	25.96	94.38	0.76	0	0.76	1.62

Tabla 5: Resultados globales en el PAR con 10 % de restricciones

	Iris				Ecoli				Rand			
	Tasa_C	Tasa_Inf	Agr.	T	Tasa_C	Tasa_Inf	Agr.	T	Tasa_C	Tasa_Inf	Agr.	T
COKPMN	0.67	0	x	0.03	36.51	632.80	x	1.16	0.76	0	x	0.03
BL	0.67	0	0.67	2.85	22.99	189.20	25.53	153.13	0.76	0	0.76	2.42

Tabla 6: Resultados globales en el PAR con 20 % de restricciones

Como he podido comprobar, el greedy converge muy rapido en los datasets iris y rand. El algoritmo BL también pero con mas coste computacional.

En donde se ha demostrado el potencial de encontrar una solución válida y en la que se minimiza la función objetivo(desviación e infeasibility) ha sido en el dataset ecoli. Al ser este mas complejo que iris y rand, BL encuentra soluciones mucho mejores que el greedy.

Hice pruebas cambiando el valor del parámetro lambda, multiplicando el valor de la máxima distancia del dataset por distintos valores. Estas las hice con ecoli ya que con los otros dos no tiene sentido, ya que converge muy rápido sin necesidad de ajustes para lambda.

De la siguiente forma:

$$f = \overrightarrow{C} + (infeasability * \lambda * f)$$

Para simular una distancia mayor a la máxima.

f = 1

General deviation: 23.025432543188337 Infeasibility: 88 lambda: 0.0268295466782898 Agr: 25.38643265087784 Elapsed Time: 141.58519864082336

f = 2

General deviation: 23.06640666682124 Infeasibility: 62 lambda: 0.0536590933565796 Agr: 26.393270454929176 Elapsed Time: 153.0076458454132

f = 3

General deviation: 24.67574305752793 Infeasibility: 42 lambda: 0.0804886400348694 Agr: 28.056265938992446 Elapsed Time: 134.59658646583557

f = 10

General deviation: 26.302439943521172 Infeasibility: 24 lambda: 0.26829546678289795 Agr: 32.74153114631072 Elapsed Time: 97.1479868888855

Se puede apreciar como cuanto mayor relevancia le demos a la infeasibility, más se descuida la desviacion. Por lo que depende del tipo de solución que uno desee puede ajustar lambda a su criterio.

Y he dejado en la carpeta graficas algunos graficos que he obtenido en dos dimensiones de los datasets simples.

6. Manual de uso

Para poder ejecutar la practica se utilizaría el siguiente comando:

$./par\_class.py < dataset >< numeroderestricciones >< algoritmoausar >$