UNIVERSIDAD DE GRANADA

Grupo 1 – Miercoles 17.30 - 19.30

Metaheurísticas — Técnicas de Búsqueda Local y Algoritmos Greedy para el Problema del Agrupamiento con Restricciones

Juan Mota Martínez
juanmotam@correo.ugr.es

Marzo 23, 2020

Índice general

§0.1: Introducción al problema de Agrupamiento por restricciones

Consiste en la agrupación de diversas instancias de un conjunto de datos, en una partición de k-clusters, de forma que se minimize la desviación general y se cumpla con el mayor número de restricciones dadas por los datos, dichas restricciones pueden ser ML(han de estar en el mismo cluster) y CL (han de ir en clusters separados).

Para resolver este problema se van a empelar los siguientes algoritmos: Búsqueda Local y K-medias.

§0.2: Representación de los datos

Se ha optado por crear dos clases Cluster y ARP, para la representación y almacenamiento de los datos, a continuación su especificación:

§0.2.1: Cluster

La clase cluster, como su propio nombre indica representa los propios clusters de datos, está compuesta por:

- **Centroide**, se trata da un vector n-dimensional de floats.
- **Datos**, un set de enteros que almacena los índices que pertenecen al centroide

§0.2.2: ARP

Se trata de la clase principal en la que se almacenan los datos, restricciones y el conjunto de clusters, posee diversas variables y métodos para proporcionar un funcionamiento correcto de los algoritmos pero las variables que nos interesan a nosotros son las siguientes:

- **datos**, un vector de vectores de floats que almacena los puntos n-dimensionales.
- **restricciones**, un map que emplea como clave un pair de enteros que simbolizan los datos y la restricción asignada, cuando se lee el fichero de restricciones se ignoran aquellas restricciones que no sean 1 o -1.
- **solucion** un vector de Clusters que usaremos para realizar las comprobaciones pertienetes además de mostrar por pantalla las distintas asignaciones

Código de la función para caluclar la Infactivilidad Total

```
Infactividad:
   errores = 0

   Para toda restric in restricciones:
      //de esta forma nos aseguramos de que sólo recorremos la matriz superior
      if(restric.indice1 < restric.indice2):
         cluster1 = buscaCluster(restric.indice1)
         cluster2 = buscaCluster(restric.indice2)

         if(cluster1 == cluster2 && restric.tipo == -1):
            errores++
         if else(cluster1 != cluster2 && restric.tipo == 1):
            errores++

   return errores
```

Código para Desviación General

```
DesviacionGeneral():
   Para todo clust in Clusters:
      Para todo indice in clust.Datos():
         distancia += abs(clust.centroide - indice.datos)
      distancia/clust.numeroDatos()

   return distancia/numeroClusters
```

§0.3: Algoritmo K-medias

K-medias aporta un enfoque greedy para la resolución del problema, se recorren aleatoriamente los datos y se asignan a un cluster seleccionando primero aquel que incumpla el menor número de restricciones, si varios de ellos comparten este número, se seleccionará aquel cuyo centroide sea más cercano al dato en concreto.

Tras esto se calcula la infactividad de la solución y se compara con la infactividad anterior, en caso de ser similares (una diferencia de 0.005 o menos) se considera que se ha alcanzado un mínimo local y se acepta el resultado. En caso contrario, se actualizan los centroides, se vacían los clusters y se vuelve a realizar la asignación.

Código de la función para calcular la infactividad generada

```
funcion calculaInfactividadGenerada(dato, cluster):

   Para toda restric in restricciones:
      if (restric.indice1 == dato):
         cluster2 = devuelveCluster(restric.indice2)

         if (cluster2 == cluster && restric.tipo == -1):
            errores++
         if (cluster2 != cluster && restric.tipo == 1):
            errores++

   return errores
```

Código del algoritmo k-medias

```
kmedias(semilla):

   desordenarAleatoriamente(indices)

   while(continuar):
```

```
6
7        Para todo indice in indices:
8           Para todo clust in Clusters:
9
10             if(calculaInfactivlidadGenerada(indice, clust) < inficativilidadminima):
11                infactivilidadminima = calculaInfactivlidadGenerada(indice, clust)
12
13             if(distanciaminima < calculaDistancia(clust.centroide, datos[indice]) &&
               infactivlidadminima == calcularInfactivlidadGenerada(indice, clust)):
14                //actualizamos la distanciaminima
15                clusteraniade = aniade
16
17
18          introduceDato(clusteraniade, indice)
19
20       actualizarCentroides()
21
22       //comprobamos si la nueva valoración es igual a la anterior, en caso positivo
         terminamos y devolvemos la solución actual, en caso contrario vaciamos los clusters
          y repetimos
```

§0.4: Búsqueda Local

Este algoritmo parte de una solución generada aleatoriamente, y se evalua su vecindario, comprobando si la función objetivo mejora o no, en caso afirmativo, se sustituye la solución inicial por la nueva y se vuelve a empezar. Con la inteción de evitar que se produzca una ejecución excesivamente larga, se ha introducido un número máximo de evaluaciones (100000), también si se recorre todo el vecindario y no encuentra ninguna mejora se asume que se ha alcanzado un mínimo local y se devuelve la solución.

No se ha implementado una función objetivo como tal, simplemente se realizan las comprobaciones pertienentes durante la propia ejecución del algoritmo, dichas comprobaciones tienen la siguiente forma:

función Objetivo

```
1  funcionObjetivo(dato, clusterIni, clusterNuevo):
2
3     restricciones -= calcularErrorGenerado(dato, clusterIni)
4     restricciones += calcularErroGenerado(dato, clusterNuevo)
5
6     intercambiaDato(dato, clusterIni, clusterNuevo)
7     //En caso de que la función objetivo no nos devuelva un valor mejor que el anterior
      este cambio se revierte
8
9     valoracion += calcularDistanciaIntraCluster() + restricciones*lambda
10
11    return valoracion
```

Búsqueda Local

```
1  BL(semilla):
2
3     AsignarIndicesClusters()
4
5     valoracionIni = calcularDistanciaIntraCluster + calcularInfactivilidad*lambda
6
7     Para todo indice in indices:
8
9        valoracion = valoracionIni
10
```

```
11        cluster = generaClusterAleatorio
12        clusterIni = buscaCluster(indice)
13
14        valoracion = funcionObjetivo(indice, clusterIni, cluster)
15
16        if(valoracion < valoracionIni):
17           Reiniciamos la búsqueda del vecindario con la nueva solucion
18
19     //Si recorremos todo el vecindario sin encontrar una solución mejor  o alcanzamos las
       100000 evaluaciones paramos y devolvemos la solución actual
```

§0.5: Tablas y análisis de los resultados

Greedy con 10 % restricciones

	Iris				Ecoli				Rand			
	Tasa_C	Tasa_inf	Agr.	T	Tasa_C	Tasa_inf	Agr.	T	Tasa_C	Tasa_inf	Agr.	T
Ejecución 1	0,24	15,00	4,31	0,14	9,06	1427,00	7,65	5,80	0,30	210,00	9,30	0,14
Ejecución 2	0,41	48,00	1,02	0,10	6,92	1306,00	669,49	7,66	0,23	194,00	8,55	0,17
Ejecución 3	0,24	40,00	0,75	0,11	8,75	1305,00	699,78	5,67	0,20	191,00	8,38	0,14
Ejecución 4	0,24	68,00	1,11	0,19	10,51	1029,00	555,39	8,15	0,19	79,00	3,57	0,14
Ejecución 5	0,24	21,00	0,51	0,20	7,46	1357,00	726,03	5,48	0,85	156,00	7,53	0,19
Media	0,27	38,40	1,54	0,15	8,54	1284,80	531,67	6,55	0,35	166,00	7,47	0,16

Greedy con 20 % restricciones

	Iris				Ecoli				Rand			
	Tasa_C	Tasa_inf	Agr.	T	Tasa_C	Tasa_inf	Agr.	T	Tasa_C	Tasa_inf	Agr.	T
Ejecución 1	0,24	75,00	0,73	0,28	10,38	2624,00	715,67	18,06	0,25	295,00	6,77	0,19
Ejecución 2	0,43	98,00	1,08	0,21	9,09	2254,00	614,93	14,34	0,19	213,00	4,90	0,30
Ejecución 3	0,38	21,00	0,24	0,22	7,37	2124,00	578,28	14,72	0,21	44,00	1,18	0,26
Ejecución 4	0,24	101,00	0,90	0,34	12,31	1844,00	507,95	17,95	0,20	23,00	0,71	0,29
Ejecución 5	0,23	31,00	0,44	0,33	7,74	2850,00	773,78	10,06	0,87	126,00	3,66	0,29
Media	0,30	65,20	0,68	0,28	9,38	2339,20	638,12	15,03	0,35	140,20	3,44	0,27

BL con 10 % restricciones

	Iris				Ecoli				Rand			
	Tasa_C	Tasa_inf	Agr,	T	Tasa_C	Tasa_inf	Agr,	T	Tasa_C	Tasa_inf	Agr,	T
Ejecución 1	0,41	62,00	1,20	0,96	6,23	815,00	437,79	24,26	0,29	101,00	4,61	24,89
Ejecución 2	0,31	234,00	3,30	0,50	6,19	712,00	383,22	71,60	0,26	136,00	6,09	72,17
Ejecución 3	0,42	192,00	2,88	1,03	6,12	545,00	294,71	35,85	0,22	90,00	4,08	36,25
Ejecución 4	0,45	112,00	1,88	0,39	6,18	656,00	353,55	50,05	0,18	59,00	2,71	50,39
Ejecución 5	0,45	133,00	2,15	1,28	6,58	424,00	231,10	46,39	0,19	48,00	2,25	46,69
Media	0,41	146,60	2,28	0,83	6,26	630,40	340,07	45,63	0,23	86,80	3,95	46,08

BL con 20 % restricciones

	Iris				Ecoli				Rand			
	Tasa_C	Tasa_inf	Agr,	T	Tasa_C	Tasa_inf	Agr,	T	Tasa_C	Tasa_inf	Agr,	T
Ejecución 1	0,22	499,00	3,52	0,78	5,85	1042,00	285,93	63,91	0,11	153,00	3,49	0,60
Ejecución 2	0,26	437,00	3,15	0,71	5,93	1066,00	292,45	65,30	0,13	122,00	2,83	0,67
Ejecución 3	0,33	463,00	3,39	0,95	3,10	654,00	178,89	926,00	0,23	82,00	2,04	0,52
Ejecución 4	0,32	461,00	3,37	1,25	7,05	816,00	226,38	531,52	0,09	240,00	5,40	0,45
Ejecución 5	0,31	460,00	3,352	1,37	5,82	1345,00	367,34	80,01	0,22	61,00	1,57	0,43
Media	0,29	464,00	3,36	1,01	5,55	984,60	270,20	333,35	0,16	131,60	3,07	0,53

Comparación Bl y k-medias 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
COPKM	0,27	38,40	1,54	0,15	8,54	1284,80	531,67	6,55	0,35	166,00	7,47	0,16
BL	0,41	146,60	2,28	0,83	6,26	630,40	340,07	45,63	0,23	86,80	3,95	46,08

Comparación BL y k-medias 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
COPKM	0,30	65,20	0,68	0,28	9,38	2339,20	638,12	15,03	0,35	140,20	3,44	0,27
BL	0,29	464,00	3,36	1,01	5,55	984,60	270,20	333,35	0,16	131,60	3,07	0,53

En primer lugar se puede observar que k-medias encuentra soluciones mucho más rápido que BL, auque esto viene también parcialmente condicionado por las semillas empleadas para la generación aleatoria de datos. Salvo para iris, BL encuentra mejores soluciones que greedy, además, de presentar una menor infactivilidad, sin embargo el resultado del experimento es el esperado, BL proporiona una mejores soluciones que k-medias en un tiempo mayor.

§0.6: Guía de uso

Para compilar el programa basta con ejecutar la órden **make** y dentro de la carpeta bin, llamar a los programas ./greedy y ./bl, es necesario que los ejecutables y los ficheros de restricciones se encuentren en el mismo directorio, cada ejecutable está preparado con las semillas seleccionadas para repetir las pruebas realizadas, e imprimirá por pantalla los clusters generados, la infactivilidad, tiempo, desviación media y agregado de cada solución obtenida. Los ficheros practica1.cpp y practica2.cpp corresponden al código fuente de greedy y búsqueda local respectivamente.

Se ha añadido también otro programa llamado prueba al que es posible suministrar argumentos para realizar los test pertinentes, el modo de uso es: ./prueba <fichero de datos><fichero de restricciones><numero de clusters><semilla><greedy>. si no se introduce greedy como argumento final ejecuta una búsqueda local.