Universidad de Granada

E.T.S Ingeniería Informática y de Telecomunicación
Grado en Ingeniería Informática
Departamento de Ciencias de la Computación e Inteligencia Artificial

Metaheurísticas - MH
Grupo A1 (Miércoles 17:30 - 19:30)
Curso 2019/2020

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

Javier Rodríguez Rodríguez
78306251Z
e.doblerodriguez@go.ugr.es

Índice

1. Descripción del problema

El problema propuesto consiste en el agrupamiento (clustering) de instancias de datos no etiquetadas que están sujetos a un conjunto de restricciones de instancia de tipo Must-Link (ML) o Cannot-Link (CL). Estas restricciones se consideran débiles, es decir, no son limitantes a una solución, pero el cumplimiento de estas es un factor importante de calidad de la solución.

Este problema, perteneciente al campo de aprendizaje semi-supervisado, es NP-Completo, y por tanto, resulta imposible obtener un algoritmo que halle la solución óptima en un tiempo razonable. Entendiendo esta limitación, y razonando dentro del campo de la metaheurística, se propone un algoritmo de Búsqueda Local (BL) con estrategia "el primer mejor" como metaheurística apropiada para resolver el problema de forma eficiente. Con el fin de comprobar tal proposición, se establece la necesidad de un segundo algoritmo, no fundamentado en metaheurísticas, que sirva para compararlo. Para este segundo algoritmo, se propone un algoritmo greedy, el K-medias con restricciones (COPKM).

La comparación entre ambos algoritmos se establecerá en dos niveles distintos: la calidad de la solución y la eficiencia temporal. La calidad debe medir adecuadamente qué tan parecidos son realmente los datos que se agrupan en un mismo cluster y cuántas restricciones incumple, mientras que la eficiencia temporal debe medir el tiempo requerido para dar una solución que el criterio del algoritmo considere suficiente.

2. Aplicación de los algoritmos:

Para aplicar los algoritmos descritos, primero que nada, es necesario describir las estructuras de datos que contienen toda la información. Los datos de entrada, las instancias y las restricciones, son provistas en forma de archivos en texto plano cuyos nombres identifican el conjunto de datos de donde se extrajeron, y en el caso de ser restricciones, el porcentaje de relaciones restringidas. Estos ficheros se leen y almacenan en una estructura de matriz: para las instancias, cada fila corresponde a una instancia, mientras que las columnas corresponden a las distintas dimensiones de cada instancia; para las restricciones, es una matriz cuadrada simétrica donde cada valor [i,j] $\in \{-1,0,1\}$ corresponde a la restricción entre la instancia i y la instancia j. Cada valor representa un tipo de restricción, siendo -1 una restricción CL, 1 una restricción ML y 0 ninguna restricción.

La solución del problema, para un conjunto de instancias y restricciones dado, vendrá dado en un vector de tantos elementos como instancias de datos haya, y que en cada posición contendrá el índice del cluster al que pertenece la instancia en dicha fila en la matriz de datos: es decir, Solución[i] contendrá el número de cluster al que pertenece Datos[i].

Para la medición de la calidad se emplea una función objetivo, cuyo valor ha de expresar la tasa de diferencia entre los datos de un mismo cluster y la cantidad de restricciones incumplidas, y por tanto, se ha de minimizar mientras de mayor calidad sea la solución. Utilizaremos la siguiente función

$$\textbf{Objetivo} = C_{General} + \textbf{Infeasibility} * \lambda$$

donde:

- $C_{General}$ corresponde a la desviación general de la solución, es decir, el valor medio de desviación de cada cluster. La desviación de un cluster corresponde al valor medio de las distancias entre su centroide asociado y cada una de las instancias contenidas en el cluster. Es decir:

$$(1)\ C_{General} = \sum_{i=1}^{k} C_i / k$$

$$(2)\ C_i = \sum_{j=1}^{ni} |\mu_i - x_j| / j$$

  donde k sea la cantidad de clústeres, μ_i el centroide del cluster i, x_j el dato j en el cluster y ni el número de datos en el cluster i.

- Infeasibility es un número entero que corresponde a la cantidad de restricciones incumplidas

- λ es un factor de escala, con el propósito de darle peso suficiente a la infeasibility y que se valore correctamente respecto al valor de $C_{General}$. Un valor general que se considera correcto es un valor mayor a la distancia máxima entre un par de instancias, seleccionando nosotros el techo de dicha distancia, entre la cantidad de restricciones. Es decir:

$$(3)\ \lambda = \text{techo}(D_{max}) / |R|$$

siendo $|R|$ el número de restricciones y D_{max} la distancia máxima entre dos instancias. Importante destacar, de cara a la implementación, que en la contabilización de restricciones debemos evitar contabilizar cada restricción dos veces (debido a la simetría de la matriz)

Para la medición de la eficiencia temporal, consideraremos como métrica el tiempo de ejecución de ambos algoritmos en condiciones similares (mismo equipo bajo carga similar).

Por último, es necesario establecer, dado que ambos algoritmos contienen instrucciones estocásticas, es necesario especificarle a los algoritmos un generador de números aleatorios que, para una semilla dada, sea capaz de replicar los valores generados.

3. Métodos de búsqueda:

Para el problema dado se establece un único método de búsqueda, el algoritmo de Búsqueda Local (BL) o Local Search. Este algoritmo plantea una estrategia "el primer mejor" para recorrer el vecindario de una solución dada, seleccionando la primera que minimice la función objetivo respecto al valor para la solución en ese instante, momento en el cual se toma el vecino como solución y se repite el proceso. Este ciclo continúa hasta que se ha explorado todo el vecindario de una solución sin encontrar un vecino que aporte un valor de la función objetivo menor.

De esta descripción se desprenden tres segmentos principales del algoritmo: el establecimiento de una solución inicial y su vecindario, la exploración de dicho vecindario y la comparación de funciones objetivo, y el cambio en la solución una vez se encuentra un vecino mejor (o la rutina de salida, en caso de no existir).

```
Búsqueda Local (datos, restricciones, k, λ, random):
      # Segmento de inicialización: establecemos solución aleatoria y calculamos centroides y función objetivo
      solución = Barajar([i_0, i_1, …, i_n-k-1] ∪ [0, 1, …, k-1]) con i_j ∈ [0,k-1]
      centroides = []
      para i = 0:k
            centroides[i] = Media(datos[x_j == i])
      dist_intracluster = []
      para  i = 0:k
             dist_intracluster[i] = Media(|centroides[i] - datos[j : solución[j] == i]|)
      infeasibility = 0
      para i = 0:n
            datos_ML = [j < i : restricciones[i][j] == 1]
            datos_CL = [j < i : restricciones[i][j] == -1]
            infeasibility += Longitud([j : solución[i] != solución[datos_ML[j]])
                  + Longitud([j : solución[i] == solución[datos_CL[j]])
      c_general = Media(dist_intracluster)
      objetivo = c_general_1 + (infeasibility * λ)

      # Segmento de generación de vecinos
      mientras no sea mejor solución:
           elementos_singleton = [i : ∄ j tal que solución[i] == solución[j]]
           elementos_posibles = [0..n] \ elementos_singleton
              vecindario = Shuffle[[i, j] : i ∈ elementos_posibles, j ∈ [0..k] \ solución[j]]

      # Segmento de exploración de vecinos
            para índice, cluster ∈ vecindario:
             nueva_sol = solución
             nueva_sol[índice] = cluster
             nuevos_centroides = centroides
             nuevas_dintracluster = dist_intracluster
             datos_ML = [j : restricciones[índice][j] == 1]
             datos_CL = [j : restricciones[índice][j] == -1]
```

```
            aporte_inf_original = Longitud([i:solución[índice] != solución[j ∈
datos_ML]) + Longitud([i : solución[índice] == solución[j ∈ datos_CL])

aporte_inf_nuevo=Longitud([i:nueva_sol[índice]!=nueva_sol[j∈datos_ML])
            + Longitud([i : nueva_sol[índice] == nueva_sol[j ∈ datos_CL])
             nueva_infeas = infeasibility - aporte_inf_original + aporte_inf_nuevo
             cluster_original = solución[índice]
             nuevos_centroides[cluster] = Media(datos[j:x_j == cluster])
             nueva_dintracluster[cluster] = Media(|nuevos_centroides[cluster]
            - datos[j : solución[j] == cluster]|)
             nuevos_centroides[cluster_original]=
Media(datos[j:x_j==cluster_original])
             nueva_dintracluster[cluster]=
Media(|nuevos_centroides[cluster_original]
            - datos[j : solución[j] == cluster_original]|)
             nueva_C = Media(nueva_dintracluster)
             nuevo_obj = nueva_C + nueva_infeas * λ
             si nuevo_obj < objetivo:
                       objetivo = nuevo_obj
                       solucion = nueva_sol
                       centroides = nuevos_centroides
                       dist_intracluster = nuevas_dintracluster
                       infeasibility = nueva_infeas
                       c_general = nueva_C
                       Salir_for
      Retornar c_general, infeasibility, objetivo
```

Este pseudocódigo describe a nivel de implementación el siguiente esquema de diseño:

1) Se establece una solución inicial aleatoria y se calculan los valores asociados a ella. En la solución inicial aleatoria se introduce un vector con todos los posibles clústeres y luego se baraja, esto para evitar que algún cluster quede vacío.
2) Inicia un ciclo cuya única condición de parada es no encontrar una mejor solución. En cada iteración de este ciclo generamos un vecindario virtual, esto es, un conjunto de pares ordenados [i, j], donde $i \in [0,n-1]$ refiere a una posición de la solución y $j \in [0,k-1]$ refiere a un cluster. Importante restringir en este vecindario virtual aquellos elementos de la solución que sean el único elemento en su cluster (pues el vecino sería inválido, ya que al cambiar el elemento dejaría el cluster vacío) y aquellos vecinos que sean iguales a la solución (mover un elemento al cluster en el que ya se encuentra).
3) Recorre cada vecino válido, generando una copia de todos los elementos que forman parte de la solución y la función objetivo, actualizándose acorde al cambio propuesto por el vecino y comparando las métricas. Si la solución del vecino da un mejor valor de la función objetivo, se actualizan los valores y se sale de la exploración, para generar un nuevo vecindario. Si no se consigue ningún vecino mejor, se indica así al ciclo externo para finalizar la ejecución del algoritmo.

4) Algoritmos de comparación:

Se optó por utilizar un único algoritmo de comparación, el k-medias con restricciones débiles (COPKM), un algoritmo Greedy en el que se explora el universo de soluciones elemento por elemento, seleccionando en cada iteración el mejor cluster disponible para el elemento. Su ejecución se puede separar en tres segmentos principales: la creación de clústeres aleatorios, la selección de cluster para cada elemento y la actualización de centroides, siendo estos últimos dos segmentos los que componen el ciclo y la mayor parte de la ejecución.

```
k-medias restringido (datos, restricciones, k, λ, random):
      # Generación de centroides aleatorios
      mínimos = [Min(datos[i][0]), …, Min(datos[i][d-1])]
      máximos = [Max(datos[i][0]), …, Max(datos[i][d-1])]
      centroides = [Random(mínimos[0], máximos[0]), …, Random(mínimos[d-1],
máximos[d-1])
      RSI = Barajar([0,1,...,n-1])
      centroides_anteriores = []
      mientras centroides != centroides_anteriores:
         solución = [-1,...,-1]
         infeasibility = 0
         # Construcción de mejor solución
         para i ∈ RSI:
             asignados = [j : solución[j] != -1]
             datos_ML = [j ∈ asignados : restricciones[i][j] == 1]
             datos_CL = [j ∈ asignados : restricciones[i][j] == -1]
             infeas_cluster = [0₀,...,0ₖ]
             infeas_cluster[j : j ∉ datos_ML] += Sum(datos_ML)
             infeas_cluster[j : j ∈ datos_CL] += Sum(datos_CL)
             mínima_infeas = Min(infeas_cluster)
             infeasibility += minima_infeas
             clústeres_considerados = [j : infeas_cluster[j] == minima_infeas]
             si Longitud(clústeres_considerados) > 1:
                    distancias = [|datos[i]-centroides[j:j∈
      clústeres_considerados]
                    cluster_seleccionado = 1er i : Min(distancias) == distancias[i]
             si no:
                    cluster_seleccionado = clusteres_considerados[0]
             solución[i] = cluster_seleccionado
         para i = 0:k:
             centroides[i] = Media(datos[j:solución[j] == i])
         si centroides_anteriores != centroides:
             centroides_anteriores = centroides
      # Ya no hay mejora
      para i=0:k:
          distancia_intracluster[i]=Media(|centroides[i] - datos[j:solución[j]==i]
      c_general = Media(distancia_intracluster)
      objetivo = c_general + infeasibility * λ
      Retornar c_general, infeasibility, objetivo
```

Este pseudocódigo describe el siguiente esquema:

1) Se crean centroides aleatorios, acotados entre los valores mínimos y máximos de los datos para cada dimensión. Se aleatoriza el orden en el que se recorren los datos.
2) Mientras los haya cambio en los centroides, se calcula para cada dato cuánta infeasibility generaría en cada cluster al que se pudiese asignar, y se seleccionan aquellos en los que sea mínima.
3) Si solo hay un posible cluster, se asigna a ese directamente. Si no, calculamos la distancia del dato a cada uno de los centroides de los clústeres considerados, y se asigna a aquel con menor distancia. En caso de haber más de un cluster equidistante, se asigna al primero según el orden en el vector de centroides.
4) Se recalculan los centroides, esta vez mediante la distancia media entre cada uno de los datos asignados a cada cluster. Se compara si estos nuevos centroides son iguales a los anteriores. Si no lo son, se reemplazan los anteriores por los actuales y se continúa la iteración. Si son iguales, se calculan los datos necesarios para el cálculo de la función objetivo y se retornan.

5) Procedimiento para desarrollar la práctica:

Todo el proceso de procesamiento de datos, implementación de algoritmos y obtención, cálculo y almacenamiento de los resultados finales fue hecho en el lenguaje de programación Python 3.7.4, a través de su distribución especializada para ciencia de datos, Anaconda. La razón de la elección de este lenguaje es por ser uno de los más utilizados y destacados en problemas de aprendizaje, por lo que se consideró una buena oportunidad para practicar su uso y sintaxis, y en particular, el uso relacionado al paquete de cómputo científico NumPy, una de las principales ventajas del lenguaje contra sus competidores y la principal herramienta detrás de todas las implementaciones.

Las ejecuciones fueron realizadas sobre una distribución Ubuntu 18.04 contenida en Windows Subsystem for Linux (WSL), en un equipo con sistema operativo Windows 10 1909.

El código proporcionado se decidió dejar de lado, pues no se consideró necesario implementar el código en Python cuando NumPy provee las herramientas de randomización necesarias.

El proceso de desarrollo de la práctica se hizo de forma lineal: primero se construyeron las estructuras de datos y mecanismos de lectura de los datos, después se implementó el algoritmo Greedy k-medias con restricciones débiles y por último el algoritmo de Búsqueda Local. Tras esto se depuraron exhaustivamente ambos y se construyó el mecanismo de salida de los datos.

- **Manual de ejecución:**

Es requisito necesario para la ejecución de la práctica que el equipo donde se ejecuten contenga Python 3.4 en adelante, así como el paquete NumPy 1.17. Además, el fichero P01b.py debe estar en el mismo directorio que un directorio llamado “Instancias y Tablas PAR 2019-20”, que contenga los ficheros de datos y restricciones. Estos ficheros deben ser los contenidos en el fichero comprimido “nuevos conjuntos de datos PAR 2019-20.zip” disponible en la plataforma PRADO.

Una vez se tengan todos los requisitos, tan solo es necesario indicar por terminal la ejecución del script, mediante la sintaxis

> <path a Python> P01b.py

Durante la ejecución no imprimirá información por pantalla, pues todos los resultados estarán contenidos en el fichero “solutions_P01b.txt” que se creará en el mismo directorio que contiene el script.

6) Experimentos y análisis de resultados:

Una ejecución del programa siguiendo las instrucciones anteriores genera la siguientes tablas de resultados. En todos los casos las semillas correspondientes a cada ejecución son 1, 112, 241, 27, 472, y el tiempo viene expresado en segundos.

Resultados obtenidos por el algoritmo Búsqueda Local en el PAR con 10% de restricciones

	Iris				Ecoli			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0.67	0.00	0.67	0.30	21.27	103.00	24.03	4.65
Ejecución 2	0.67	0.00	0.67	0.36	21.71	70.00	23.59	5.41
Ejecución 3	0.67	0.00	0.67	0.36	22.43	75.00	24.44	6.20
Ejecución 4	0.67	0.00	0.67	0.27	19.26	163.00	23.63	4.71
Ejecución 5	0.67	0.00	0.67	0.38	21.62	87.00	23.95	4.40
Media	0.67	0.00	0.67	0.33	21.26	99.60	23.93	5.07

	Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0.72	0.00	0.72	0.25	10.81	113.00	14.99	0.63
Ejecución 2	0.72	0.00	0.72	0.30	10.82	115.00	15.07	0.64
Ejecución 3	0.72	0.00	0.72	0.30	13.83	6.00	14.06	0.55
Ejecución 4	0.72	0.00	0.72	0.28	13.83	6.00	14.06	0.63
Ejecución 5	0.72	0.00	0.72	0.26	10.82	108.00	14.81	0.46
Media	0.72	0.00	0.72	0.28	12.02	69.60	14.60	0.59

Resultados obtenidos por el algoritmo Greedy COPKM en el PAR con 10% de restricciones

	Iris				Ecoli			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0.67	11.00	0.75	0.02	39.87	373.00	49.87	1.86
Ejecución 2	0.67	4.00	0.69	0.02	36.31	374.00	46.34	0.16
Ejecución 3	0.67	0.00	0.67	0.02	37.54	291.00	45.35	3.39
Ejecución 4	0.67	7.00	0.72	0.02	39.26	341.00	48.41	0.14
Ejecuión 5	0.67	0.00	0.67	0.02	35.16	77.00	37.22	0.12
Media	0.67	4.40	0.70	0.02	37.63	291.20	45.44	1.13

	Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0.72	0.00	0.72	0.02	15.35	53.00	17.31	0.03
Ejecución 2	0.71	9.00	0.78	0.02	16.48	101.00	20.21	0.04
Ejecución 3	0.72	0.00	0.72	0.03	14.15	54.00	16.14	0.04
Ejecución 4	0.72	0.00	0.72	0.02	13.83	70.00	16.41	0.04
Ejecuión 5	0.72	0.00	0.72	0.01	15.12	114.00	19.33	0.03
Media	0.71	1.80	0.73	0.02	14.98	78.40	17.88	0.04

Resultados obtenidos por el algoritmo Búsqueda Local en el PAR con 20% de restricciones

	Iris				Ecoli			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0.67	0.00	0.67	0.32	21.57	259.00	25.04	4.00
Ejecución 2	0.67	0.00	0.67	0.32	21.94	168.00	24.20	3.86
Ejecución 3	0.67	0.00	0.67	0.40	21.96	162.00	24.14	3.71
Ejecución 4	0.67	0.00	0.67	0.33	21.94	134.00	23.74	4.35
Ejecuión 5	0.67	0.00	0.67	0.35	21.83	179.00	24.23	4.72
Media	0.67	0.00	0.67	0.34	21.85	180.40	24.27	4.13

	Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0.72	0.00	0.72	0.30	10.86	231.00	15.12	0.45
Ejecución 2	0.72	0.00	0.72	0.27	14.29	0.00	14.29	0.52
Ejecución 3	0.72	0.00	0.72	0.30	10.81	257.00	15.56	0.64
Ejecución 4	0.72	0.00	0.72	0.26	10.87	235.00	15.21	0.60
Ejecuión 5	0.72	0.00	0.72	0.29	14.29	0.00	14.29	0.51
Media	0.72	0.00	0.72	0.28	12.22	144.60	14.89	0.54

Resultados obtenidos por el algoritmo Greedy COPKM en el PAR con 20% de restricciones

	Iris				Ecoli			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0.67	10.00	0.71	0.03	39.73	268.00	43.32	0.14
Ejecución 2	0.67	0.00	0.67	0.02	34.75	86.00	35.90	0.18
Ejecución 3	0.67	21.00	0.74	0.02	36.36	315.00	40.58	0.12
Ejecución 4	0.67	0.00	0.67	0.02	38.99	262.00	42.50	0.99
Ejecuión 5	0.67	0.00	0.67	0.03	36.75	183.00	39.20	0.08
Media	0.67	6.20	0.69	0.02	37.31	222.80	40.30	0.30

	Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0.72	0.00	0.72	0.02	14.29	0.00	14.29	0.03
Ejecución 2	0.72	0.00	0.72	0.02	14.10	76.00	15.51	0.03
Ejecución 3	0.72	0.00	0.72	0.02	15.20	567.00	25.68	0.03
Ejecución 4	0.72	0.00	0.72	0.01	14.78	106.00	16.74	0.03
Ejecuión 5	0.72	0.00	0.72	0.01	14.18	102.00	16.07	0.04
Media	0.72	0.00	0.72	0.02	14.51	170.20	17.66	0.03

Resultados globales en el PAR con 10% de restricciones

	Iris				Ecoli			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
COPKM	0.67	4.40	0.70	0.02	37.63	291.20	45.44	1.13
BL	0.67	0.00	0.67	0.33	21.26	99.60	23.93	5.07

	Rand				Newthyroid			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
COPKM	0.71	1.80	0.73	0.02	14.98	78.40	17.88	0.04
BL	0.72	0.00	0.72	0.28	12.02	69.60	14.60	0.59

Resultados globales en el PAR con 20% de restricciones

	Iris				Ecoli			
	Tasa_C	***Tasa_inf***	***Agr.***	**T**	***Tasa_C***	***Tasa_inf***	***Agr.***	**T**
COPKM	0.67	6.20	0.69	0.02	37.31	222.80	40.30	0.30
BL	0.67	0.00	0.67	0.34	21.85	180.40	24.27	4.13

	Rand				Newthyroid			
	Tasa_C	***Tasa_inf***	***Agr.***	**T**	***Tasa_C***	***Tasa_inf***	***Agr.***	**T**
COPKM	0.72	0.00	0.72	0.02	14.51	170.20	17.66	0.03
BL	0.72	0.00	0.72	0.28	12.22	144.60	14.89	0.54

Analizar los resultados obtenidos nos proporciona mucha información sobre la naturaleza de los conjuntos de datos y la calidad de los algoritmos. Sobre los conjuntos de datos, resulta evidente que Ecoli es un set de mayor complejidad que el resto, pues las soluciones dan valores agregados del orden de las decenas, en contraste con el resto de conjuntos, cuyos valores agregados no alcanzan el orden de la unidad, y el tiempo de ejecución de los algoritmos para este conjunto está en el orden de los segundos, en contraste con los milisegundos en los que se ejecutan para el resto de conjuntos.

De los otros conjuntos, es posible notar que Rand e Iris son considerablemente sencillos, dando en muchas ejecuciones los mismos resultados, lo que da la intuición de pensar que son óptimos (en particular porque su Tasa_inf es 0). Newthyroid es un punto intermedio de complejidad, con agregados mucho más elevados que los de Iris y Rand, que no superan la unidad, pero en tiempo de ejecución comparable a éstos, en contraste con Ecoli que junto con su incremento en Agregado incrementa el tiempo de ejecución.

Sobre los algoritmos, es evidente que el COPKM, en su naturaleza Greedy, resulta mucho más eficiente en tiempo, con un tiempo medio de ejecución por debajo de los 5 milisegundos en 3 de los 4 conjuntos de datos, y en el Ecoli, que ya fue notado como el más complejo en cuanto a tiempo de ejecución, le toma en promedio menos de un segundo. Sin embargo, el beneficio en cuanto a tiempo de ejecución es contrastado con una desmejora en la calidad de la solución, llegando a soluciones peores en la mayoría de los casos. Sin embargo, para casos sencillos y con cuidado de realizar suficientes iteraciones, resulta una herramienta poderosa para conjuntos sencillos: en Iris y Rand, alcanza sus mejores ejecuciones los mejores valores alcanzados con Búsqueda Local, pero carece de la consistencia para llegar a ellos en todas las ejecuciones. Esta debilidad puede compensarse si se realizan múltiples ejecuciones y se conserva el mejor valor, proceso que no representa demasiado costo por su ya destacada eficiencia temporal.

El algoritmo de Búsqueda Local, por su parte, es mucho más poderoso y consistente. Si bien el tiempo de ejecución es del orden de decenas de milisegundos en todos los conjuntos excepto Ecoli, y en general tomando entre 5 y 10 veces más tiempo que COPKM (con la mayor diferencia en Newthyroid 10 y 20 y Ecoli 20, donde es casi 15 veces más lento), la consistencia y calidad de las soluciones es sin duda mejor. Para Iris y Rand alcanza la misma solución en todas las ejecuciones, dando a intuir que posiblemente sea la óptima y llegando a ella en todos los casos, y en Newthyroid y Ecoli ofrece valores de Infeasibility y agregado notablemente mejores, siendo cerca de la mitad del agregado provisto por COPKM para Ecoli. Esta diferencia notable indica sin duda alguna que la calidad de Búsqueda Local lo convierte en un algoritmo mucho más apto para conjuntos de datos de mayor complejidad donde difícilmente la naturaleza Greedy de COPKM lo apunte a valores cercanos al óptimo.

7) Referencias bibliográficas:

- https://numpy.org/doc/1.17/
- Seminario 2. Problemas de optimización con técnicas basadas en búsqueda local. Óscar Cordón García.
- https://docs.python.org/3/library/pathlib.html
- Constrained K-means Clustering with Background Knowledge. Proceedings of the Eighteenth International Conference on Machine Learning, 2001, p. 577–584. Kiri Wagstaff, Claire Cardie, Seth Rogers, Stefan Schroedl.
- https://realpython.com/python-timer/
- http://zetcode.com/python/fstring/