PRÁCTICA 3

TÉCNICAS DE BÚSQUEDA BASADA EN TRAYECTORIAS

PARA EL PROBLEMA DE AGRUPAMIENTO CON RESTRICCIONES (PAR)

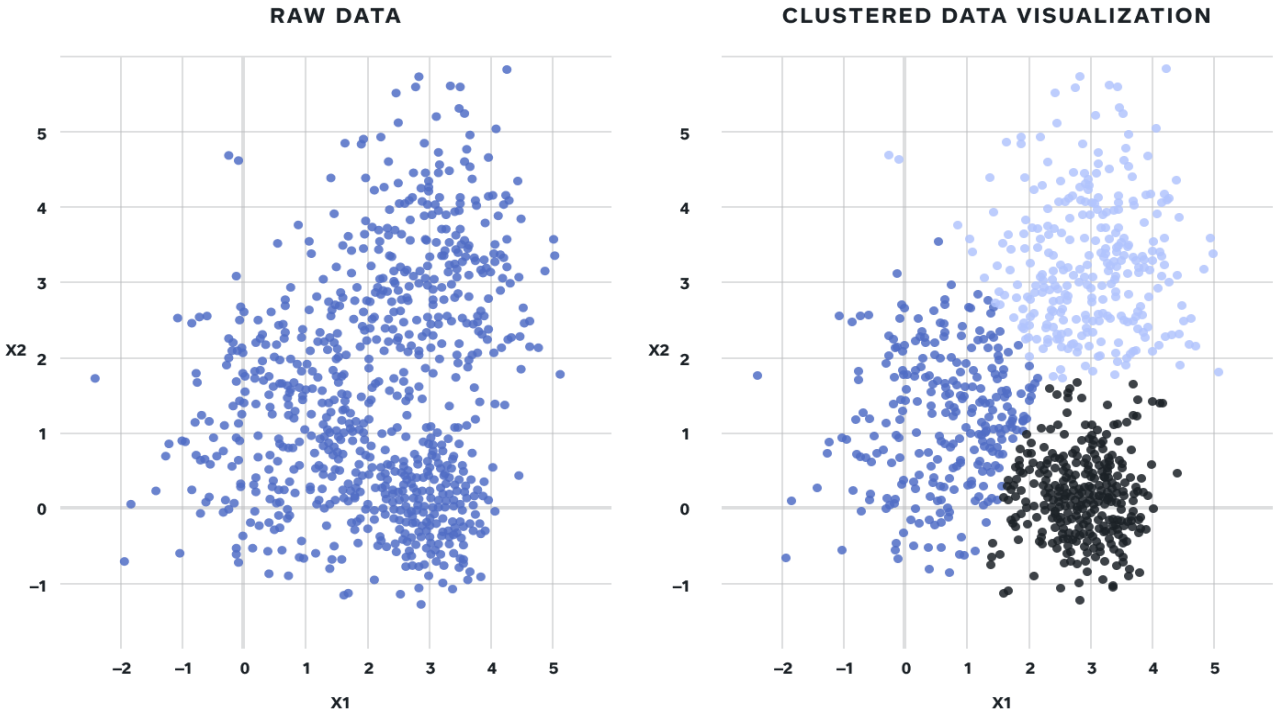
Antonio José Blánquez Pérez
3ºCSI METAHEURÍSTICAS
45926869D
ajose99bp@correo.ugr.es
Grupo 2 – Jueves

ÍNDICE

1. DESCRIPCIÓN DEL PROBLEMA....3
2. FORMALIZACIÓN Y DESCRIPCIÓN DE LA APLICACIÓN DE LOS ALGORITMOS....4
2.1 FORMALIZACIÓN DE CONCEPTOS....4
2.2 REPRESENTACIÓN DE LA INFORMACIÓN....5
2.3 FUNCIÓN OBJETIVO....5
2.4 DATAFRAMES....7
3. DESCRIPCIÓN DE METAHEURÍSTICAS....8
3.1 ALGORITMO DE BÚSQUEDA LOCAL EL PRIMER MEJOR....8
3.2 BMB....9
3.3 ILS....9
3.4 ENFRIAMIENTO SIMULADO....10
3.5 ILS-ES....11
4. DESCRIPCIÓN DEL ALGORITMO DE COMPARACIÓN COPKM....12
5. PROCEDIMIENTO CONSIDERADO PARA EL DESARROLLO DE LA PRÁCTICA....14
5.1 CÓDIGO Y LIBRERÍAS USADAS....14
5.2 MANUAL DE USUARIO....14
6. ANÁLISIS DE RENDIMIENTO....15
6.1 DESCRIPCIÓN DE LOS CASOS DEL PROBLEMA....15
6.2 RESULTADOS OBTENIDOS....15
6.3 ANÁLISIS DE RESULTADOS....17

1. DESCRIPCIÓN DEL PROBLEMA

El agrupamiento o clustering busca clasificar objetos de un conjunto en subconjuntos o clusters a través de sus posibles similaridades. Es una técnica de aprendizaje no supervisado que permite descubrir grupos inicialmente desconocidos o agrupar objetos similares, de manera que podemos encontrar patrones en grandes grupos de datos que serían imposibles(o extremadamente difíciles) de encontrar sin esta técnica. Una muestra de agrupamiento sería la siguiente:



Un ejemplo cotidiano del problema del agrupamiento sería dividir una cantidad de frutas en verdes, poco maduras y muy maduras que, aunque trivial, nos permite entender el concepto que hay detrás de la técnica que nos ocupa. Con ésto podemos comprender otras aplicaciones del clustering, como pueden ser el análisis de caracteres escritos a mano, muestras de diálogo, huellas dactilares o imágenes, la clasificación de especies en subespecies o el agrupamiento para moléculas o proteínas.

Para poder aplicar esta técnica debemos medir d características de nuestro grupo de n objetos, por ejemplo el color o la textura en el caso de las frutas o la simetría o la intensidad en el caso de caracteres escritos a mano, obteniendo una conjunto de datos de longitud n con d dimensiones.

En nuestro caso concreto abordaremos una variación del clustering clásico, el Problema de Agrupamiento con Restricciones(PAR), es decir, además de nuestro conjunto de datos tenemos cierta información a la que llamaremos restricciones o, más concretamente, restricciones de instancia. Ésto quiere decir que tenemos alguna información sobre objetos que tienen que pertenecer al mismo cluster(ML, Must-Link) y sobre objetos que no pertenecen al mismo cluster(CL, Cannot-Link), siendo éstas restricciones débiles, o lo que es lo mismo, podemos incumplir restricciones pero debemos minimizar el número de incumplidas al máximo.

El planteamiento combinatorio de este problema es NP-Completo, por tanto a continuación propondremos algunas alternativas de algoritmos para intentar resolver el problema de forma aproximada en un tiempo razonable. El problema se abordará con k(nº de clusters) conocida, ya que su búsqueda es otro problema complejo.

2. FORMALIZACIÓN Y DESCRIPCIÓN DE LA APLICACIÓN DE LOS ALGORITMOS

2.1 FORMALIZACIÓN DE CONCEPTOS

Vamos a definir distintos conceptos que se nombrarán a lo largo del documento y que son necesarios para entender los procedimientos en cada algoritmo.

Primeramente podemos formalizar la definición de nuestro conjunto de datos como una matriz X de n·d, n objetos en un espacio de d dimensiones(el número de medidas que tenemos sobre cada objeto). Lo podemos notar matemáticamente como:

$$\vec{x_i}=\{x_{[i,1]},...,x_{i,d}\}/x_{[i,d]}\in\mathbb{R}\ \forall\ j\in\{1,..,d\}$$

Llamaremos $C=\{c_1,\ldots,c_k\}$ al conjunto de los k clusters, de manera que cada c_i será un subconjunto de X y podrá asociada una etiqueta l_i que lo nombre Para cada cluster es posible calcular su centroide asociado $\vec{\mu}_i$, siendo el vector promedio de sus instancias, de la siguiente manera:

$$\vec{\mu}_i=\frac{1}{|c_i|}\sum_{\vec{x_j}\in c_i}\vec{x_j}$$

También debemos definir la distancia media intra-cluster, $\overline{c_i}$, como la media de las distancias entre cada instancia del cluster y su centroide asociado, usando en este caso la distancia euclídea, aunque es equivalente a usar, por ejemplo, la distancia Manhattan. Es calculable con la siguiente expresión:

$$\overline{c_i}=\frac{1}{|c_i|}\sum_{\vec{x_j}\in c_i}\|\vec{x_j}-\vec{\mu}_i\|_2$$

A través de estas, podemos llegar a la desviación general de C, la media de las desviaciones intra-cluster, que nos será de gran ayuda para minimizar la solución y para el posterior análisis de la solución. La expresión sería la siguiente:

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

Por otro lado tenemos las restricciones, $R=ML\cup CL$, donde $ML(\vec{xi},\vec{xj})$ indica que las instancias $\vec{x}_i\, y\, \vec{x}_j$ deben estar asignadas al mismo cluster y $CL(\vec{x}_i,\vec{x}_j)$ que las instancias $(\vec{x}_i,\vec{x}_j)$ no pueden ser asignadas al mismo cluster. Con ello, definimos la infactibilidad o infeasibility, que es un indicador de las restricciones que incumple nuestra solución, como:

$$infeasibility=\sum_{i=0}^{|ML|}1\left(h_c(\vec{ML_{[i,1]}})\neq h_c(\vec{ML_{[i,2]}})\right)+\sum_{i=0}^{|CL|}1\left(h_c(\vec{CL_{[i,1]}})=h_c(\vec{CL_{[i,2]}})\right)$$

Siendo 1 la función booleana que devolverá 1 si la expresión que toma como argumento es verdadera y 0 en otro caso.

2.2 REPRESENTACIÓN DE LA INFORMACIÓN

Ahora que tenemos algunos conceptos claros y formalizados, pasamos a ver como representaremos la información necesaria en el proceso de resolución del problema.

Primero aclarar que los datos de entrada, nuestro dataframe, estará organizado en una matriz implementada como un vector<vector<float> >(ambos vectores de la librería STL), es decir, un vector de longitud n de vectores de reales en coma flotante de longitud d.

Para las restricciones usaremos dos formas de representación, una para facilitar el acceso cuando conocemos la restricción que queremos comprobar y otra para cuando queramos recorrer todas las restricciones. Para el primer caso se usará la misma representación que para el dataframe, mientras que para el segundo se construirá una lista con elementos del tipo [x,y,{1,-1}], siendo x e y los dos elementos que tiene la restricción y el último elemento 1 si conforman una restricción ML y -1 si es CL. Esta lista se ha implementado usando un vector<vector<int> >, también de la STL. De ésta manera, al recorrer esa lista evadiremos todos los pares que no tienen una restricción y nuestra búsqueda será mucho más eficiente que en una matriz.

Por último, representaremos la solución con un vector<int> de longitud n, de manera que la posición i contendrá el número del cluster asignado al objeto i.

2.3 FUNCIÓN OBJETIVO

La función objetivo es la función que debemos minimizar, es decir, la función que dice como de buena es la solución que le pasamos como parámetro.

De todas las definiciones que hemos hecho, desviación general($\overline{C}$) e infactibilidad(infeasibility) nos dan información acerca de la bondad de la solución, la primera a través de la distancia promedio de cada dato a su centroide correspondiente y la segunda a través de la medida del incumplimiento de las restricciones de la solución dada. Ambas han de ser minimizadas, pero hay que darle más o menos importancia a una y a otra, de manera que debemos introducir un parámetro con el que controlar ésta importancia que debatiremos posteriormente. Así, nuestra función objetivo queda definida como:

$$f = \overline{C} + (infeasibility) * \lambda$$

Dicho ésto queda definir el parámetro λ, el cual se ha decidido proponer como el entero superior a la distancia máxima D multiplicada por un parámetro rel(relevancia) y dividida por el número de restricciones totales R.

$$\lambda = \frac{\lceil D \rceil * rel}{|R|}$$

Posteriormente, en el punto 6, se hará un estudio de λ en función del parámetro rel.

A continuación se expone la descripción en pseudocódigo de todos los operadores y de la función objetivo.

calular_infeasibility

```
begin
    Para cada instancia de lista_restricciones
    begin
        Si ( solucion[lista_restricciones[i][0]] = solucion[lista_restricciones[i][1]] ) and
    ( lista_restricciones[2] = -1 )
            entonces infactibilidad ← infactibilidad +1

        Si ( solucion[lista_restricciones[i][0]] != solucion[lista_restricciones[i][1]] ) and
    ( lista_restricciones[2] = 1 )
            entonces infactibilidad ← infactibilidad +1
    end

    return infactibilidad
end
```

calcular_centroides

```
begin
    Para cada instancia i de solucion
        centroide[i] ← centroide[i] + dataframe[i.posicion]

    Para cada centroide
        centroide ← centroide / cluster.size

    return centroides
end
```

calcular_distancia_media_intracluster

```
begin
    Para cada instancia i de solucion
    begin
        distancia_intracluster[i] ← distancia_intracluster[i] +
    distancia_euclidea(centroide[i],dataframe[i.posicion])
    end

    Para cada cluster
        distancia_intracluster[cluster] ← distancia_intracluster[cluster] / cluster.size
    return distancia_intracluster
end
```

calcular_desv_general
begin
    Para cada cluster
        desv_general ← desv_general + distancia_intracluster[cluster]

    desv_general ← desv_general / #clusters

    return desv_general
end

calcular_lamda
begin
    dist_max ← calcular_máxima_distancia(dataframe)
    lambda ← dist_max * rel / lista_restricciones.size

    return lambda
end

calcular_f_objetivo
begin
    inf ← calcular_infeasibility()
    dist_intra ← calcular_distancia_intracluster()
    desv ← calcular_desv_general()
    lamb ← calcular_lambda()

    return desv + inf * lamb
end

2.4 DATAFRAMES

Los conjuntos de datos usados para testear la bondad de los algoritmos son los siguientes:

- Iris: Contiene información sobre las características de tres tipos de flores de Iris. Tiene 3 clases(k=3) y 6 dimensiones.

- Ecoli: Contiene medidas sobre las ciertas características de diferentes tipos de células que pueden ser empleadas para predecir la localización de ciertas proteínas. Tiene 8 clases(k=8) y 7 dimesiones.

- Rand: Conjunto de datos artificial formado por tres agrupamientos bien diferenciados generados en base a distribuciones normales. Tiene 3 clases(k=3) y 2 dimensiones.

- Newthyroid: Contiene medidas cuantitativas tomadas sobre la glándula tiroides de 215 pacientes. Tiene 3 clases(k=3) y 5 dimensiones.

3. DESCRIPCIÓN DE METAHEURÍSTICAS

3.1 ALGORITMO DE BÚSQUEDA LOCAL EL PRIMER MEJOR

Este algoritmo de búsqueda local se basa en comenzar con una solución generada aleatoriamente e ir explorando vecinos, los cuales deben ser válidos, y elegir el primero mejor. Los vecinos se exploran de manera aleatoria. A continuación se describirá el pseudocódigo de la generación de la solución aleatoria y de las dos versiones implementadas de la búsqueda y selección de vecino.

genera_sol_aleatoria
```
begin
    Hacer
    begin
        Para cada i entre 0 y dataframe.size
            solucion[i] ← rand() % k
    end
    Mientras algun cluster esté vacío
end
```

escoger_primer_mejor_vecino_v1
```
begin
    aleatorio ← rand()
    Para i entre 0 y (solucion.size)*(k-1) y mientras f_mejor >= f_actual
    begin
        acceso ← (i+aleatorio)%c.size
        solucion[acceso] ←  (solucion[acceso] + (i/solucion.size) + 1) % k

        f_actual ← calcular_f_objetivo(solucion)
    end
end
```

escoger_primer_mejor_vecino_v2
```
begin
    Para cada posicion i de la solucion y para cada cluster j
        vecinos.add([ i, (solucion[i] + j)%k ])

    vecinos.shuffle

    Para cada vecino y mientras f_mejor >= f_actual
        f_actual ← calcular_f_objetivo(vecino)
end
```

busqueda_local
begin
    vecino ← genera_sol_aleatoria
    f_actual ← calcular_f_objetivo(vecino)
    Mientras f_actual mejore o no hayamos llegado a 100000 evaluaciones
        escoger_primer_mejor_vecino_v2
end

El procedimiento de generar la solución aleatoria es más bien trivial, por lo que no requiere una explicación muy compleja, simplemente creamos el vector solución con tamaño #dataframe con el generador de números aleatorios.

La función escoger_primer_mejor_vecino se repite hasta que no se encuentre una f más pequeña o se completen 100000 evaluaciones de f.

La v1 funciona tal que: en cada iteración se accede a una posición al azar del vector y se itera secuencialmente a partir de ahí, sumando algún número entre 1 y k-1(accediendo de manera secuencial a todos los vecinos) dependiendo del azar. La v2 simplemente genera todos los vecinos, los baraja y busca el primer mejor vecino, además funciona mejor que v1, por lo que es la que se usará para la comparación entre algoritmos.

3.2 BMB

El algoritmo de búsqueda local multiarranque(BMB) es simple, es exactamente igual al de búsqueda local, exceptuando que en lugar de hacer las 100000 evaluaciones sobre la misma solución se crean 10 soluciones aleatorias y se aplican 10000 evaluaciones a cada una, introduciendo así un componente de exploración. El pseudocódigo quedaría así:

bmb
begin
    Repetir 10 veces
        solucion[i] ← generar_sol_aleatoria
    Para cada solución
        busqueda_local(con límite de 10000 evaluaciones de f)
    Devolver la solución con menor f de las 10
end

3.3 ILS

La búsqueda local iterativa o Iterated Local Search(ILS) es también una variación de BL, de tal manera que cada 10000 evaluaciones de las 100000 se muta un valor de la solución actual, otra manera de introducir exploración. El pseudocódigo también es sencillo.

mutar(s)
being
    indice ← numero aleatorio entre 0 y tamaño de s
    s[indice] = (s[indice] + aleatorio entre 1 y k-1) % k
end

ils
begin
    s ← generar_sol_aleatoria
    Repetir 10 veces
        sp ← s
        mutar(s)
        busqueda_local(con límite de 10000 evaluaciones de f)
        Si sp es mejor que s
            s ← sp
end

3.4 ENFRIAMIENTO SIMULADO

Este es el algoritmo más complicado de esta práctica, paso a explicar detalladamente cada parte.

Se han ajustado los valores relacionados con el algoritmo, dando los siguientes los mejores resultados:

$T_{ini} = \frac{0.3 * f_{ini}}{-\log(0.3)}$, T_{fin} = 0.001 , max_vecino = 8*sol_inicial.size , max_exitos = 0.1*max_vecinos

Se han probado dos esquemas de enfriamiento, optando finalmente por el primero:

cauchy_cooling_scheme(T)
begin
    M = 100000 / max_vecinos
    beta = (T_ini – T_fin) / (T_ini*T_fin*M)
    Devolver T / (1+beta*T)
end

proportional_cooling_scheme(T)
begin
    Devolver T*alpha
end

Se ha necesitado usar una función que genere un aleatorio real entre 0 y 1, ya que todas las funciones anteriormente usadas solo generaban enteros. Esta función está recogida de internet, ya que tuve bastantes problemas en este punto y la solución final fue esta. La función es la siguiente:

fRand(min,max)
begin
    f = (double)rand() / RAND_MAX
    Devolver min + f*(max-min)
end

Como digo, tras un gran número de pruebas, estos han sido los parámetros que mejores resultados han dado, por tanto son los que se establecen al ejecutar el algoritmo.

Dicho todo esto, el algoritmo ha quedado de la siguiente manera:

enfriamiento_simulado
```
begin
    s ← generar_sol_aleatoria
    f_ini ← calcular_f_objetivo(s)
    Hasta que T_ini > T_fin
        T_fin /= 10
    Mientras evaluaciones<100000 y exitos>0 y T>=T_fin
    begin
        exitos ← 0
        Mientras vecinos<max_vecinos y exitos<max_exitos
        begin
            sp ← s
            mutar(sp)
            f_act ← calcular_f_objetivo(sp)
            delta ← f_act – f_ini
            Si delta<0 o fRand(0,1)<=exp(-delta/T)
                s ← sp
                f_ini = f_act
                exitos++
        end
        T = cauchy_cooling_scheme(T)
    end
    Devolver mejor solución encontrada
end
```

El único cambio respecto a las indicaciones que se han dado ha sido el de bajar un 20% el número máximo de vecinos para así hacer el enfriamiento algo más lento, permitiendo así una mejor exploración que no se daba con el valor anterior.

3.5 ILS-ES

Finalmente ILS-ES es una hibridación de ILS y ES, sustituyendo BL por ES en ILS, quedando así:

ils-es
```
begin
    s ← generar_sol_aleatoria
    Repetir 10 veces
        mutar(s)
        enfriamiento_simulado(con límite de 10000 evaluaciones)
    Devolver la mejor solución encontrada
end
```

4. DESCRIPCIÓN DEL ALGORITMO DE COMPARACIÓN COPKM

El algoritmo de comparación COPKM, mediante una política greedy, está basado en el algoritmo k-medias, pero adaptado al uso de restricciones, de manera que lo que minimiza en cada inserción es el aumento de la infactibilidad. Se han implementado dos variantes, se detallan más adelante las diferencias. La descripción de ambas en pseudocódigo sería la siguiente:

copkm_v1

```
begin
    indices ← (0,1,…,dataframe.size -1)
    indices.shuffle
    Para cada cluster
        Generar centroide con coordenadas aleatorio en el rango del dataframe

    Mientras cambie la asignación de cluster
    begin
        Para cada objeto i en el dataframe
        begin
            i2 ← indices[i.posicion]
            Para cada objeto j en solucion
            begin
                Si restricciones[i2][j] = -1
                    entonces infactibilidad[solucion[j]]++
                Si restricciones[i2][j] = 1
                    entonces para cada cluster m != solucion[j] infactibilidad[m]++
            end

            min ← cluster con menor infactibilidad, desempatando con la mínima distancia
            solucion.add(min)
        end

        copia_sol ← solucion
        Para cada i entre 0 y la longitud de la solucion
            solucion[indices[i]] ← copia_sol[i]

        calcular_centroides()
    end
end
```

copkm_v2

```
begin
    indices ← (0,1,…,dataframe.size -1)
    indices.shuffle
    Para cada cluster
        Generar centroide con coordenadas aleatorio en el rango del dataframe

    Mientras cambie la asignación de cluster
    begin
```

```
        Para cada objeto i en el dataframe
        begin
            i2 ← indices[i.posicion]
            Si es primera vez, entonces para cada objeto j en solucion
            begin
                Si restricciones[i2][j] = -1
                    entonces infactibilidad[solucion[j]]++
                Si restricciones[i2][j] = 1
                    entonces para cada cluster m != solucion[j] infactibilidad[m]++
            end

            Si no, para cada cluster j
            begin
                solucion[i2] = j
                infactibilidad[j] = calcular_infactibilidad(solucion)
            end0

            min ← cluster con menor infactibilidad, desempatando con la mínima distancia
            Si es primera vez, solucion.add(min)
            Si no, c[i2] = min
        end

        Si es primera vez, entonces
        begin
            copia_sol ← solucion
            Para cada i entre 0 y la longitud de la solucion
                solucion[indices[i]] ← copia_sol[i]
        end

        calcular_centroides()
    end
end
```

La v1 es exactamente lo que dice el pseudocódigo, para cada iteración va introduciendo los objetos(en un orden aleatorio) en el cluster que sume menor infactibilidad, se recalculan los centroides en base a esa solución y se vuelve a generar otra solución con los clusters en su nueva posición.

La v2 es tan solo una versión algo mejorada de la v1, aunque la mejora cambia totalmente el resultado como veremos posteriormente. La mejora es básicamente que en cada iteración no se empieza de nuevo, si no que se tiene en cuenta la solución anterior, de manera que al introducir los objetos(en un orden aleatorio) se calcula la infactibilidad de cada cluster conforme a la solución anterior con el cambio de cluster del objeto. Esta v2 tiene un parecido razonable con BL, sin embargo, en ningún sitio se nos dice que no se pueda guardar la solución anterior, por lo que, aunque compararé ambas versiones, he mantenido ésta para las comparativas con otros algoritmos.

5. PROCEDIMIENTO CONSIDERADO PARA EL DESARROLLO DE LA PRÁCTICA

5.1 CÓDIGO Y LIBRERÍAS USADAS

La práctica ha sido desarrollada en C++, siendo todo el código implementado a mano, con ayuda de las explicaciones dadas en clase y en el guión, y usando las siguientes librerías:

- iostream: para entrada y salida por teclado y pantalla respectivamente.
- fstream: para entrada y salida por ficheros.
- STL: todos los vectores usados provienen de ésta librería.
- cmath: para algunos cálculos necesarios los algoritmos.
- algorithm: para usar la función shuffle()
- ctime: para medir tiempos de ejecución
- cstdlib: para la generación de números aleatorios

5.2 MANUAL DE USUARIO

Compilación:

Se incluye un makefile, por lo que se compila tan solo ejecutando la orden ‘make’ desde la terminal.

Ejecución completa:
La ejecución completa planteada para el ejercicio(8 conjuntos(4*2 % de restricciones)*5 ejecuciones distintas) se pueden hacer con ayuda del script incluido. Tan solo es necesario darle permisos de ejecución y ejecutarlo con el algoritmo como parámetro, ‘copkm’ para el algoritmo de comparación COPKM, ‘bl’ para el algoritmo de búsqueda local primero el mejor, ‘aggun’ para AGG-UN, ‘aggsf’ para AGG-SF, ‘ageun’ para AGE-UN, ‘agesf’ para AGE-SF. ‘am1’ para AM(10,1), ‘am0.1’ para AM(10,0.1), ‘am0.1mej’ para AM(10,0.1mej), ‘bmb’ para BMB, ‘ils’ para ILS, ‘es’ para ES o ‘ilses’ para ILS-ES.

Ejecución manual:
Si se desea ejecutar a mano un sólo conjunto, se puede hacer llamando a ‘clustering’ de la siguiente forma:

./clustering <dataframe> <restricciones> <k> <ALG> <seed>

ALG = {0:COPKM, 1:BL, 2:AGG-UN, 3:AGG-SF, 4:AGE-UN, 5:AGE-SF, 6:AM(10,1), 7:AM(10,0.1), 8:AM(10,0.1mej), 9:ES, 10:BMB, 11:ILS, 12:ILS-ES}

También es posible llamarlo sin argumentos, entonces pedirá por teclado la información necesaria.

6. ANÁLISIS DE RENDIMIENTO

6.1 DESCRIPCIÓN DE LOS CASOS DEL PROBLEMA

Se van utilizar todos los dataframes anteriormente mencionados para analizar como de buenos son los algoritmos planteados para resolver el problema del agrupamiento con restricciones, con el 10 y el 20% de restricciones en cada conjunto de datos. Los datos organizan en tablas y se analizarán posteriormente.

Para ello se han realizado con cada uno de los cuatro dataframes, Iris, Newthyroid, Ecoli y Rand, tanto con el 10% como con el 20% de restricciones, un total de 5 veces, lo que hace un total de 40 ejecuciones(4 conjuntos x 2 conjuntos de restricciones x 5 veces).

Para poder replicar el total de ejecuciones todas ellas se han realizado utilizando semillas para la generación de números aleatorios, por lo que conociendo estas semillas se puede realizar el estudio exactamente de la misma manera y con los mismos resultados. Las semillas usadas en este estudio son las siguientes:

Ejecución 1	Ejecución 2	Ejecución 3	Ejecución 4	Ejecución 5
798245613	123456789	25022020	17042026	459268694

6.2 RESULTADOS OBTENIDOS

Muy importante: Se han vuelto a realizar las pruebas con COPKM y BL debido a un error que ha afectado al resto de prácticas de la asignatura. Las razones, el procedimiento realizado y las consecuencias de ello se especifican en el Anexo I.

Los resultados se muestran a continuación en el formato indicado:

COPKM 10%	Iris				Newthyroid				Ecoli				Rand			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0,669	0	0,669	0,000	14,29	0	14,29	0,02	39,270	105	40,240	0,125	0,715	0	0,715	0,015
Ejecución 2	0,669	0	0,669	0,000	14,29	0	14,29	0,02	36,350	29	36,617	0,328	0,715	0	0,715	0,015
Ejecución 3	0,669	0	0,669	0,000	14,29	0	14,29	0,02	38,293	6	38,348	0,218	0,715	0	0,715	0,000
Ejecución 4	0,669	0	0,669	0,000	14,29	0	14,29	0,00	34,077	0	34,077	0,375	0,715	0	0,715	0,015
Ejecución 5	0,669	0	0,669	0,015	18,46	47	19,25	0,01	33,593	3	33,621	0,296	0,715	0	0,715	0,000
Media	0,67	0,00	0,67	0,00	15,12	9,40	15,28	0,01	36,32	28,60	36,58	0,27	0,72	0,00	0,72	0,01

* El rojo representa un ciclo, su valor es la media del resto

COPKM 20%	Iris				Newthyroid				Ecoli				Rand			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0,669	0	0,669	0,015	14,29	0	14,29	0,03	28,397	0	28,397	0,359	0,715	0	0,715	0,015
Ejecución 2	0,669	0	0,669	0,015	14,29	0	14,29	0,03	33,133	0	33,133	0,375	0,715	0	0,715	0,000
Ejecución 3	0,669	0	0,669	0,015	14,29	0	14,29	0,03	33,080	3	33,093	0,265	0,715	0	0,715	0,000
Ejecución 4	0,669	0	0,669	0,000	14,29	0	14,29	0,03	32,525	2	32,534	0,390	0,715	0	0,715	0,015
Ejecución 5	0,669	0	0,669	0,015	14,29	0	14,29	0,04	28,397	0	28,397	0,390	0,715	0	0,715	0,000
Media	0,67	0,00	0,67	0,01	14,29	0,00	14,29	0,03	31,11	1,00	31,11	0,36	0,72	0,00	0,72	0,01

BL 10%	Iris				Newthyroid				Ecoli				Rand			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0,669	0	0,669	0,015	13,55	15	13,80	0,06	22,832	884	30,984	0,451	0,715	0	0,715	0,015
Ejecución 2	0,669	0	0,669	0,015	10,77	151	13,29	0,63	26,951	1304	38,976	0,328	0,715	0	0,715	0,015
Ejecución 3	0,669	0	0,669	0,015	10,89	95	12,48	0,63	24,686	921	33,179	0,531	0,715	0	0,715	0,015
Ejecución 4	0,669	0	0,669	0,015	10,52	174	13,42	0,05	28,154	1132	38,590	0,343	0,715	0	0,715	0,015
Ejecución 5	0,669	0	0,669	0,031	13,55	15	13,80	0,09	27,177	971	36,132	0,390	0,715	0	0,715	0,031
Media	0,67	0,00	0,67	0,02	11,85	90,00	13,36	0,29	25,96	1042,40	35,57	0,41	0,72	0,00	0,72	0,02

BL 20%	Iris				Newthyroid				Ecoli				Rand			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0,669	0	0,669	0,031	10,76	351	13,82	0,06	27,250	2503	39,126	0,687	0,715	0	0,715	0,031
Ejecución 2	0,669	0	0,669	0,015	10,82	263	13,12	0,06	27,978	2279	38,792	0,734	0,715	0	0,715	0,015
Ejecución 3	0,669	0	0,669	0,015	10,88	293	13,44	0,06	28,825	2176	39,149	0,844	0,715	0	0,715	0,015
Ejecución 4	0,669	0	0,669	0,015	10,67	334	13,58	0,08	23,814	1595	31,381	1,281	0,715	0	0,715	0,031
Ejecución 5	0,669	0	0,669	0,031	10,82	264	13,12	0,06	25,451	1888	34,409	1,000	0,715	0	0,715	0,015
Media	0,67	0,00	0,67	0,02	10,79	301,00	13,42	0,07	26,66	2088,20	36,57	0,91	0,72	0,00	0,72	0,02

BMB 10%	Iris				Newthyroid				Ecoli				Rand			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0,669	0	0,669	0,250	10,81	116	12,75	0,64	22,207	714	28,731	3,969	0,715	0	0,715	0,234
Ejecución 2	0,669	0	0,669	0,250	10,82	108	12,62	0,63	25,617	834	33,308	2,406	0,715	0	0,715	0,219
Ejecución 3	0,669	0	0,669	0,234	10,87	97	12,50	0,61	24,701	781	31,903	4,266	0,715	0	0,715	0,218
Ejecución 4	0,669	0	0,669	0,265	10,82	108	12,62	0,66	26,003	975	34,995	3,594	0,715	0	0,715	0,234
Ejecución 5	0,669	0	0,669	0,265	10,88	97	12,50	0,64	20,865	792	28,169	3,719	0,715	0	0,715	0,203
Media	0,67	0,00	0,67	0,25	10,84	105,20	12,60	0,63	23,94	1087,40	32,05	4,50	0,72	0,00	0,72	0,22

BMB 20%	Iris				Newthyroid				Ecoli				Rand			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0,669	0	0,669	0,296	10,88	230	12,88	0,84	22,781	1609	30,415	9,844	0,715	0	0,715	0,296
Ejecución 2	0,669	0	0,669	0,281	10,81	261	13,08	0,78	24,988	2122	35,056	8,797	0,715	0	0,715	0,281
Ejecución 3	0,669	0	0,669	0,250	10,87	235	12,92	0,69	26,101	1641	33,887	9,094	0,715	0	0,715	0,281
Ejecución 4	0,669	0	0,669	0,313	10,82	253	13,03	0,72	23,540	1374	30,059	10,703	0,715	0	0,715	0,281
Ejecución 5	0,669	0	0,669	0,281	10,81	263	13,11	0,80	20,576	1465	37,527	10,016	0,715	0	0,715	0,281
Media	0,67	0,00	0,67	0,28	10,84	248,40	13,00	0,77	23,60	1642,20	33,39	9,69	0,72	0,00	0,72	0,28

ILS 10%	Iris				Newthyroid				Ecoli				Rand			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0,669	0	0,669	0,203	10,81	111	12,66	0,33	18,855	652	24,868	2,093	0,715	0	0,715	0,156
Ejecución 2	0,669	0	0,669	0,125	13,55	15	13,80	0,44	21,680	595	27,167	1,578	0,715	0	0,715	0,125
Ejecución 3	0,669	0	0,669	0,140	13,55	15	13,80	0,44	21,631	557	26,768	1,188	0,715	0	0,715	0,140
Ejecución 4	0,669	0	0,669	0,156	10,87	98	12,51	0,33	21,254	538	26,216	1,343	0,715	0	0,715	0,140
Ejecución 5	0,669	0	0,669	0,156	13,55	15	13,80	0,45	24,951	600	30,485	1,250	0,715	0	0,715	0,140
Media	0,67	0,00	0,67	0,16	12,47	50,80	13,31	0,40	21,67	588,40	27,10	1,49	0,72	0,00	0,72	0,14

ILS 20%	Iris				Newthyroid				Ecoli				Rand			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0,669	0	0,669	0,187	10,88	230	12,88	0,45	21,227	1266	27,234	4,609	0,715	0	0,715	0,156
Ejecución 2	0,669	0	0,669	0,187	10,82	262	13,10	0,33	24,253	1270	30,280	4,156	0,715	0	0,715	0,187
Ejecución 3	0,669	0	0,669	0,203	10,82	256	13,05	0,34	21,580	1807	30,154	4,359	0,715	0	0,715	0,172
Ejecución 4	0,669	0	0,669	0,156	10,81	266	13,13	0,39	23,702	1173	29,267	4,250	0,715	0	0,715	0,172
Ejecución 5	0,669	0	0,669	0,172	13,55	41	13,91	0,50	21,817	1457	28,731	4,875	0,715	0	0,715	0,172
Media	0,67	0,00	0,67	0,18	11,38	211,00	13,22	0,40	22,52	1394,60	29,13	4,45	0,72	0,00	0,72	0,17

ES 10%	Iris				Newthyroid				Ecoli				Rand			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0,669	0	0,669	0,359	10,82	105	12,57	0,20	16,280	288	18,936	1,234	0,715	0	0,715	0,156
Ejecución 2	0,669	0	0,669	0,265	13,55	15	13,80	0,23	16,206	325	19,203	1,969	0,715	0	0,715	0,141
Ejecución 3	0,669	0	0,669	0,234	10,81	108	12,62	0,13	19,202	170	20,770	2,094	0,715	0	0,715	0,141
Ejecución 4	0,669	0	0,669	0,250	13,55	15	13,80	0,38	19,210	173	20,806	1,359	0,715	0	0,715	0,094
Ejecución 5	0,669	0	0,669	0,188	10,82	115	12,74	0,14	16,264	295	18,985	2,078	0,715	0	0,715	0,109
Media	0,67	0,00	0,67	0,26	11,91	71,60	13,10	0,22	17,43	250,20	19,74	1,75	0,72	0,00	0,72	0,13

ES 20%	Iris				Newthyroid				Ecoli				Rand			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0,669	0	0,669	0,250	10,82	262	13,10	0,16	13,144	740	19,655	6,313	0,715	0	0,715	0,188
Ejecución 2	0,669	0	0,669	0,188	13,55	41	13,91	0,17	16,166	687	19,426	10,813	0,715	0	0,715	0,203
Ejecución 3	0,669	0	0,669	0,157	13,55	41	13,91	0,19	19,282	344	20,914	7,422	0,715	0	0,715	0,078
Ejecución 4	0,669	0	0,669	0,218	10,82	260	13,09	0,19	16,157	704	19,497	7,344	0,715	0	0,715	0,188
Ejecución 5	0,669	0	0,669	0,203	13,55	41	13,91	0,17	16,184	740	19,694	6,469	0,715	0	0,715	0,141
Media	0,67	0,00	0,67	0,20	12,46	129,00	13,58	0,17	16,19	643,00	19,84	7,67	0,72	0,00	0,72	0,16

ILS-ES 10%	Iris				Newthyroid				Ecoli				Rand			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0,669	0	0,669	0,609	13,55	15	13,80	0,56	18,182	573	23,467	1,594	0,715	0	0,715	0,250
Ejecución 2	0,669	0	0,669	0,297	13,55	15	13,80	0,59	17,434	454	21,621	1,578	0,715	0	0,715	0,266
Ejecución 3	0,669	0	0,669	0,297	10,90	99	12,55	0,58	19,593	360	22,913	1,578	0,715	0	0,715	0,250
Ejecución 4	0,669	0	0,669	0,266	13,55	15	13,80	0,53	18,363	500	22,974	1,594	0,715	0	0,715	0,266
Ejecución 5	0,669	0	0,669	0,281	10,81	113	12,70	0,48	17,307	548	22,361	1,563	0,715	0	0,715	0,266
Media	0,67	0,00	0,67	0,35	12,47	51,40	13,33	0,55	18,18	487,00	22,67	1,58	0,72	0,00	0,72	0,26

ILS-ES 20%	Iris				Newthyroid				Ecoli				Rand			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
Ejecución 1	0,669	0	0,669	0,328	13,55	41	13,91	0,66	19,135	904	23,424	7,656	0,715	0	0,715	0,250
Ejecución 2	0,669	0	0,669	0,343	10,82	262	13,11	0,64	20,704	552	23,323	7,328	0,715	0	0,715	0,313
Ejecución 3	0,669	0	0,669	0,359	13,55	41	13,91	0,63	20,361	790	24,109	7,391	0,715	0	0,715	0,328
Ejecución 4	0,669	0	0,669	0,375	13,55	41	13,91	0,59	17,611	867	21,725	7,703	0,715	0	0,715	0,297
Ejecución 5	0,669	0	0,669	0,359	10,82	255	13,04	0,63	16,436	981	21,090	7,500	0,715	0	0,715	0,297
Media	0,67	0,00	0,67	0,35	12,46	128,00	13,57	0,63	18,85	818,80	22,73	7,52	0,72	0,00	0,72	0,30

Esto nos deja los siguientes valores medios:

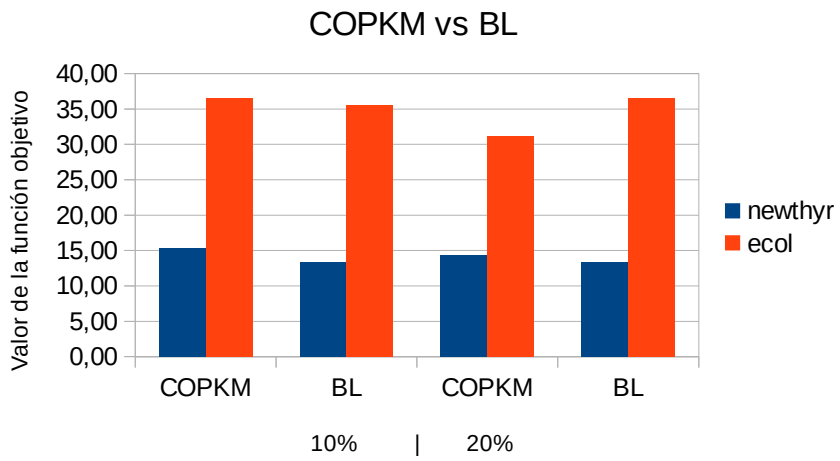
10%	Iris				Newthyroid				Ecoli				Rand			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
COPKM	0,67	0,00	0,67	0,00	15,12	9,40	15,28	0,01	36,32	28,60	36,58	0,27	0,72	0,00	0,72	0,01
BL	0,67	0,00	0,67	0,02	11,85	90,00	13,36	0,29	25,96	1042,40	35,57	0,41	0,72	0,00	0,72	0,02
BMB	0,67	0,00	0,67	0,25	10,84	105,20	12,60	0,63	23,94	1087,40	32,05	4,50	0,72	0,00	0,72	0,22
ILS	0,67	0,00	0,67	0,16	12,47	50,80	13,31	0,40	21,67	588,40	27,10	1,49	0,72	0,00	0,72	0,14
ES	0,67	0,00	0,67	0,26	11,91	71,60	13,10	0,22	17,43	250,20	19,74	1,75	0,72	0,00	0,72	0,13
ILS-ES	0,67	0,00	0,67	0,35	12,47	51,40	13,33	0,55	18,18	487,00	22,67	1,58	0,72	0,00	0,72	0,26

20%	Iris				Newthyroid				Ecoli				Rand			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
COPKM	0,67	0,00	0,67	0,01	14,29	0,00	14,29	0,03	31,11	1,00	31,11	0,36	0,72	0,00	0,72	0,01
BL	0,67	0,00	0,67	0,02	10,79	301,00	13,42	0,07	26,66	2088,20	36,57	0,91	0,72	0,00	0,72	0,02
BMB	0,67	0,00	0,67	0,28	10,84	248,40	13,00	0,77	23,60	1642,20	33,39	9,69	0,72	0,00	0,72	0,28
ILS	0,67	0,00	0,67	0,18	11,38	211,00	13,22	0,40	22,52	1394,60	29,13	4,45	0,72	0,00	0,72	0,17
ES	0,67	0,00	0,67	0,20	12,46	129,00	13,58	0,17	16,19	643,00	19,84	7,67	0,72	0,00	0,72	0,16
ILS-ES	0,67	0,00	0,67	0,35	12,46	128,00	13,57	0,63	18,85	818,80	22,73	7,52	0,72	0,00	0,72	0,30

6.3 ANÁLISIS DE RESULTADOS

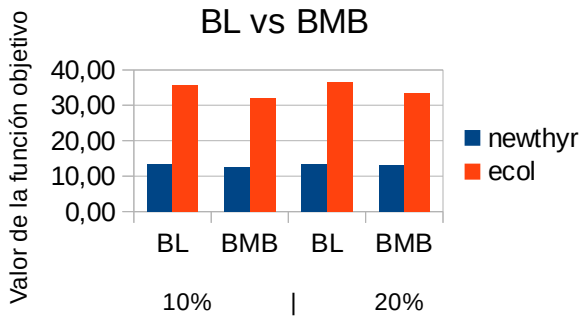
Como es obvio, todos las conclusiones que se detallan a continuación se harán teniendo en cuenta mayoritariamente los conjuntos de datos Newthyroid y Ecoli, ya que apra Iris y Rand todos los algoritmos que hemos utilizado proporcionan el óptimo.

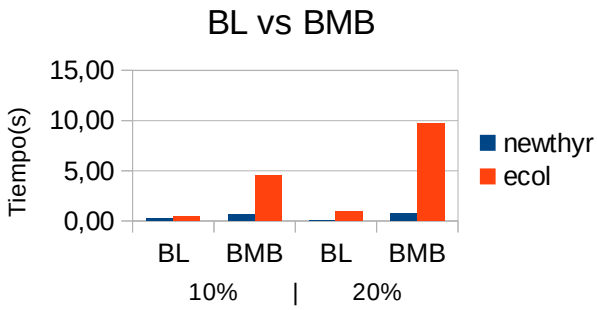
Ya que hemos vuelto a recoger pruebas con COPKM y BL, y dado que ya no ha sido necesario el uso del parámetro rel del que se habló en la práctica 1 y ahora ambas f son comparables, primero se va a volver a analizar el rendimiento de ambos.



Como vemos, COPKM funciona mejor cuanto mayor sea el número de restricciones, lo cual es lógico ya que basa su funcionamiento en ellas. A esto se debe también que, aunque saque las infactibilidades más bajas, es a costa de unas desviaciones muy altas, lo que provoca que en estos conjuntos no sea especialmente bueno obteniendo unos agregados que no compiten con el resto de algoritmos. Aun así hay una excepción en Ecoli con 20% de restricciones debido muy probablemente a que, teniendo en cuenta que Ecoli es el conjunto de datos usados que más exploración requiere como veremos posteriormente(y como ya hemos intuido en la práctica anterior), COPKM se basa en exploración(reconstruye la solución en cada iteración) y BL, aunque sea mucho mejor algoritmo en general, basa su funcionamiento en la explotación. Para concluir con COPKM podemos decir, como ya dijimos en la primera práctica, que es un gran algoritmo para conjuntos de datos sencillos debido a su corto tiempo medio de ejecución, aunque con conjuntos de datos más complejos no es suficientemente bueno.

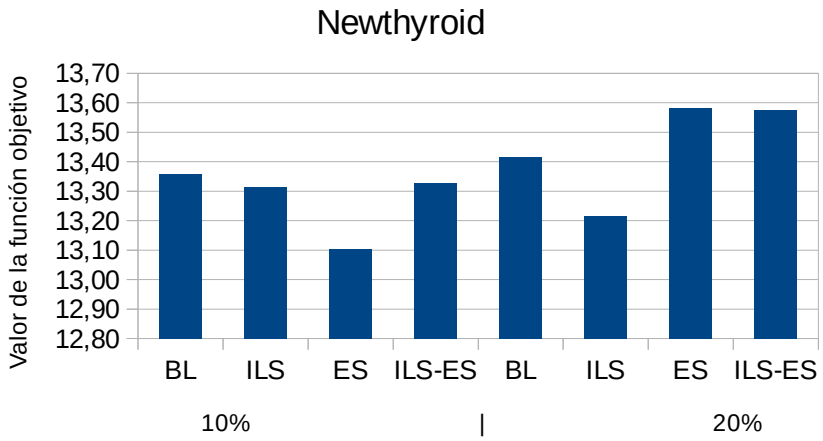
Sabiendo que en general BL es mejor que COPKM, aunque peca de centrarse en la explotación, vamos a compararla con el resto de algoritmos de esta práctica. Empezaremos por BMB.





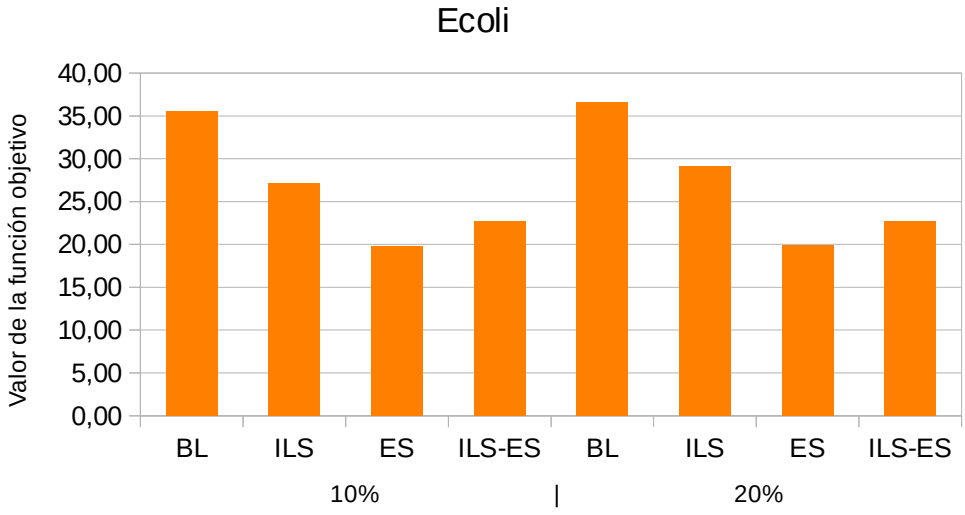
En sencillo ver en los gráficos por qué creo que BMB es un mal algoritmo, saca unos resultados pobremente mejores que BL a costa de consumir un tiempo astronómicamente mayor(de hecho es, de lejos, el algoritmo que más tiempo usa para encontrar una solución). Es por esto por lo que lo descartaría como un algoritmo competente en este contexto, por lo que, como COPKM, lo dejaré de tener en cuenta para el resto de análisis, ya que me sigue pareciendo mejor BL para las comparaciones. El único caso remarcable, sin embargo, de BMB es en Newthyroid, ya que a pesar de tardar mucho más que el resto nos da mejor solución que el resto, no obstante no creo que la diferencia sea suficiente como para tenerla en cuenta, ya que además es a costa de subir la infactibilidad en gran medida y gracias a que este conjunto de datos obtiene buenos resultados a partir de explotación.

Dicho esto podemos analizar los resultados de los cuatro algoritmos basados en trayectorias seleccionados, BL y los tres restantes.



Vemos que para Newthyroid todos son razonablemente buenos, siendo las diferencias no demasiado relevantes. Esto se debe, como ya hemos comentado, a que la exploración no es demasiado importante en Newthyroid, al menos para llegar a las soluciones que hemos conseguido a lo largo de todas las prácticas.

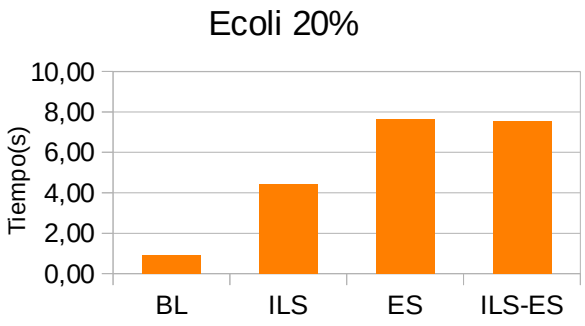
Es por ello que las diferencias entre estos algoritmos las veremos en Ecoli, donde, como ya hemos hablado, la exploración es importante para llegar a mejores soluciones. De hecho, en la práctica anterior vimos como mejoran las soluciones con un buen equilibrio entre exploración y explotación, por lo que podemos obtener conclusiones concluyentes sobre los algoritmos basados en trayectorias a partir de este estudio.

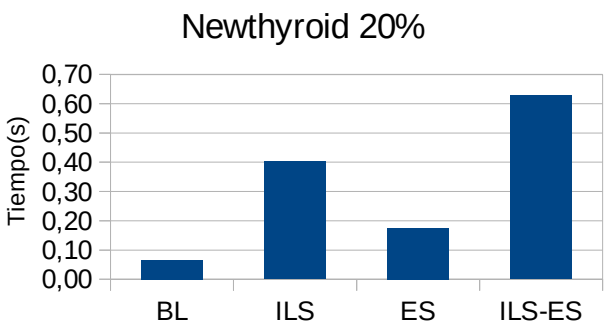


Se puede ver que ES es el mejor, obteniendo una solución mucho mejor que el resto de algoritmos. Por otra parte es lógico ver como todos mejoran en gran medida a BL, ya que BL no cuenta con ningún mecanismo de exploración, mientras que el resto consiguen hibridar de manera más o menos correcta explotación y exploración. ILS es peor que ES en mi opinión porque tiende algo más a la explotación, al final solo muta soluciones en algún momento esporádico y de manera más bien sutil.

ILS-ES, sin embargo, es mejor que ILS original y peor que ES, lo primero probablemente porque ES es mucho mejor algoritmo que BL y por tanto ILS-ES debe ser mejor que ILS. El que sea mejor ES que ILS-ES debe tener que ver con que ES ya tiene su equilibrio entre exploración y explotación y la parte de ILS lo que hace es introducir algo más de explotación llevándose por delante el gran equilibrio existente en ES ya que le resta un gran número de iteraciones.

Ahora hablemos de tiempos de ejecución:





En Ecoli tenemos los tiempos proporcionalmente parecidos que, como podemos ver en el gráfico de la derecha, son coherentes ya que a mejor solución obtenida mayor tiempo de ejecución, por supuesto con unas diferencias de tiempos asequibles(nada que ver con BMB). Sin embargo se puede ver un comportamiento extraño en Iris, Rand y Newthyroid, ya que ES es más rapido que ILS e ILS-ES. Esto puede deberse a las diferencias en el número de clusters, ya que Ecoli es el único con un número diferente, aunque no tengo muy claro el por qué puede ser.

ANEXO I: CORRECCIÓN DE ERRORES

En este anexo se va a explicar la corrección que ha sido necesaria hacer en el contexto de la práctica, lo que ha afectado a los resultados que habríamos obtenido en el resto de prácticas.

1 DETECCIÓN DEL PROBLEMA

La cuestión es que, tras obtener unos malos resultados en la práctica 2, comencé a hacer la pruebas que no pude hacer antes de la entrega anterior. El resultado fue darme cuenta de que algo no iba bien con la función objetivo, ya que no obtenía buenos resultados tampoco con los algoritmos de la práctica 3 y me dí cuenta que, con los nuevos conjuntos de datos, COPKM y BL tampoco.

Obtenida esta información comencé a hacer más pruebas que, tras mucho trabajo, no llegaron a ningún sitio. Entonces, ya sin ideas, tuve una tutoría con Óscar donde tras una hora de revisar el código no pudimos encontrar el problema que derivó en pedir a un compañero la solución óptima de algún caso para intentar encontrar el problema; esto reveló un cálculo erróneo de la infactibilidad, que devolvía un valor de casi 300 cuando debía ser 0. Tras más pruebas sin éxito obtuve el visto bueno de Óscar para pedir dicha función a un compañero para reparar el error, lo cual reparó efectivamente el código, haciendo que todos los algoritmos comenzaran a funcionar correctamente.

Óscar propuso que ante cualquier duda con este proceso puede referirse a él para comentarlas sin ningún problema, por supuesto diríjase también a mí si hubiera cualquier problema. Aclarar que debido a ésto, aunque no hay demasiado cambio, las funciones que calculan la infactibilidad y la lista de restricciones a partir de la matriz original son una adaptación de las cedidas por parte de mi compañero Diego Navarro Cabrera a mi código. No fui capaz de encontrar el error aun así, debía estar en la relación a bajo nivel entre las líneas de mi código anterior o no lo supimos ver.

2 BL Y COPKM

Por algún motivo ambos algoritmos funcionaban lo suficientemente bien como para encontrar buenas soluciones en los conjuntos de datos antiguos aún con el error, supongo que al estar ordenados. Es por este error por el que el encontré en la primera práctica mejores soluciones usando el parámetro rel(que ya no existe) para darle mas importancia a la infactibilidad, ya que era esta la que era anormalmente alta.

3 ALGORITMOS GENÉTICOS

La práctica anterior, al contrario de lo que pensaba, estaba afectada por este error y no por errores en el código de los algoritmos. No he tenido demasiado tiempo(a pesar de haberle dedicado un tiempo relativamente alto a las prácticas de esta asignatura, sobre todo comparado con el resto) para rehacer el análisis de los AGG y los AM, aunque sí que he hecho las pertinentes para poder asegurar que las conclusiones obtenidas en la práctica anterior, aún con resultados anormales, son correctas respecto al análisis del comportamiento de los algoritmos y de los conjuntos de datos.