UNIVERSIDAD
DE GRANADA

Metaheurísticas

grado en ingeniería informática

Práctica 1

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

Autor
Ignacio Vellido Expósito
ignaciove@correo.ugr.es
79056166Z

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2019-2020

Índice

1. Introducción

1.1. Descripción del problema

El Problema del Agrupamiento (PA) es un problema clásico de aprendizaje no supervisado, que consiste agrupar una serie de instancias en un número concreto de clústers de forma lógica. En estas prácticas añadimos restricciones al problema convirtiéndolo en el **Problema del Agrupamiento con Restricciones** (PAR), una variante NP-Completa semi-supervisada de PA.

En PAR por tanto se debe agrupar una serie de datos en un número predefinido de clústers, teniendo que contener cada clúster como mínimo un elemento. Además, tenemos dos tipos de restricciones, asociadas a pares de elementos:

- Must-Link (ML): Ambos elementos deben pertenecer al mismo clúster.
- Cannot-Link (CL): Ambos elementos deben pertenecer a distintos clústers.

A la hora de implementar los algoritmos se considerarán estas restricciones como débiles, es decir, serán relevantes a la hora de determinar la calidad de la solución pero no la consideración de si una posible solución lo es.

Trabajaremos con 3 instancias del problema:

1. **Iris**: Características de tres tipos de flor de Iris. Contiene 3 clases y 4 dimensiones.
2. **Ecoli**: Características de células. 8 clases y 7 dimensiones.
3. **Rand**: Conjunto de datos artificial de dos dimensiones formado por 3 clústers y 2 dimensiones.

1.2. Consideraciones previas

1.2.1. Función objetivo

La función objetivo en el problema PAR se calcula en base a la fórmula:

$$f = \overline{C} + (infeasibility * \lambda) \tag{1}$$

Siendo:

$$\lambda = \frac{\text{máx}\{d_i \in D\}}{|R|} \quad tal\ que\ D = Distancias \tag{2}$$

$$infeasibility = \sum_{i=0}^{|ML|} \mathbb{1}(h_C(\overrightarrow{ML_{[i,1]}}) \neq (h_C(\overrightarrow{ML_{[i,2]}}) + \sum_{i=0}^{|CL|} \mathbb{1}(h_C(\overrightarrow{CL_{[i,1]}}) = (h_C(\overrightarrow{CL_{[i,2]}}) \tag{3}$$

$$\overline{C} = \frac{1}{k} \sum_{c_i \in C} ||\overrightarrow{x_j} - \overrightarrow{u_j}||_2 \tag{4}$$

Que es calculada en pseudocódigo:

Algorithm 1: Función objetivo

C = Distancia media intra-cluster ;
λ = Distancia máxima en el conjunto de datos / nº de restricciones ;
inf = Nº restricciones no cumplidas ;
return *C + (λ∗ inf)*

Algorithm 2: Distancia media intra-cluster

Input: Conjunto de datos, solucion, centroides
Separar conjunto de datos según su cluster ;
for *particion del conjunto de datos* **do**
  Calcular distancia media de sus elementos al centroide correspondiente ;
end
return *C*

Algorithm 3: Infeasibility

Input: s: solución
inf = 0 ;
for *r* ***in*** *lista_restricciones* **do**
  if $r = ML$ ***and*** $s[r[0]] \neq s[r[1]]$ **then**
    inf++ ;
  else if $r = CL$ ***and*** $s[r[0]] = s[r[1]]$ **then**
    inf++ ;
end
return *inf*

1.2.2. Representación de la solución

Una solución se representa como un vector de igual longitud que el conjunto de datos, indicando en cada casilla el clúster al que pertenece el elemento i-ésimo. Adicionalmente, es necesario que cada clúster cuente como mínimo con un elemento.

2. Algoritmos

Todos los algoritmos se han implementado a mano en el lenguaje Python con la ayuda de los framework Scikit-Learn y Numpy. En algunos casos se ha seguido código de terceros (de StackOverflow) para la resolución de pequeños problemas (mejor forma de cálcular distancias, cómo generar permutaciones de arrays, etc.), y sólamente tras la comprobación de su correctitud. Adicionalmente, el cálculo de los centroides se ha hecho a partir del módulo *NearestCentroid* de Scikit.

Para la ejecución del código es necesario tener instalado Python en su versión 3, y se puede ejecutar de la siguiente manera:

```
$python main.py [-h] -a A -p P [-s S] [-ss SS]

>Argumentos:
    -h, --help  Mensaje de ayuda con esta descripcion
    -a A        Algoritmo a utilizar (copkm, bl, copkm2)
    -p P        Problema a ejecutar (iris10, ecoli20...)
    -s S        Archivo que contiene la semilla
    -ss SS      Valor de la semilla
```

Adicionalmente, es necesario tener incluido Numpy, Scikit-Learn y Matplotlib en el entorno de Python donde se ejecute.

2.1. Greedy COPKM v1

El primer algoritmo con el que afrontamos PAR es la técnica *Greedy COPKM*, variante de la versión Greedy clásica adaptada al problema PAR.

El proceso que sigue el algoritmo es el siguiente:

Algorithm 4: Algoritmo COPKM

```
Input: Conjunto de datos, lista de restricciones
solution = solución aleatoria ;
indexes = Conjunto de índices (uno por cada dato) ;
centroids = Conjunto con los centroides de cada clúster ;
Barajar indexes ;
while solution cambie do
    actualSol = solución vacía ;
    for i in indexes do
        inf = máxima infeasibility posible ;
        foreach cluster valido do
            newSol = actualSol modificando el cluster del índice i ;
            newInf = infeasibility de newSol ;
            if newInf < inf then
                Almacenar como cluster posible ;
            else if newInf < inf then
                Añadir a la lista de clusters posibles ;
        end
        distances = Distancias de los elementos a cada cluster ;
        actualSol se modifica con el cluster de los posibles con menor distancia ;
    end
    solution = actualSol ;
end
return solution, centroids
```

2.2. Búsqueda Local

La generación de vecinos se separa en dos pasos. Puesto que creamos un vecindario virtual (de forma compacta) como pares índice-cluster, al inicio del proceso BL se genera todo el posible vecindario, que corresponde a una permutación del número de elementos del problema con el número de clústers. Una vez generado esta lista inmutable de pares, en cada iteración se seleccionan aquellos que sean válidos en la solución actual. En este caso un vecino es válido cuando no deja ningún clúster vacío y no pertenece a la solución actual.

Algorithm 5: Generación de vecinos

Input: Conjunto de vecinos virtuales posibles, en forma de lista de pares, solution
vecindario = listaVecinosVirtuales ;
for *v* ***in*** *vecindario* **do**
  soluciónVecina = Solución producida aplicando el vecino *v* ;
  Eliminar *v* de *vecindario* si soluciónVecina deja algún cluster vacío o es igual que solution ;
end
return *vecindario*

La generación de soluciones aleatorias hace uso de la librería Numpy y genera un vector aleatorio relleno con posibles clústers. Este proceso se llama repetidamente hasta que la solución es válida (no deja ningún clúster vacío).

Algorithm 6: Generación de soluciones aleatorias

Input: Tamaño de la solucion, numero de clusters
do
  solution = X donde $\bigtriangledown x_i \in X, 0 \leq x_i < numClust$;
while *solution no es válida*;
return *solution*

Con estos dos algoritmos el proceso de búsqueda queda de la siguiente manera:

Algorithm 7: Proceso de búsqueda

```
Input: Conjunto de datos, lista de restricciones
solution = Solución aleatoria ;
centroids = Conjunto con los centroides de cada clúster ;
neighborhood = Permutación con todos los vecinos virtuales posibles ;
evaluations = 0 ;
cost = Valor de la función objetivo para la solución actual ;
while solution cambie and evaluations < 100 000 do
    neigh = Vecinos virtuales válidos para la solution actual ;
    for n in neigh do
        evaluations++ ;
        newSolution = Solución aplicando el vecino n ;
        newCentroids = Centroides de newSolution ;
        newCost = Valor de la función objetivo para newSolution ;
        if newCost < cost then
            cost = newCost ;
            solution = newSolution ;
            Saltar a la siguiente iteración del bucle while ;
        end
    end
end
return solution, centroids
```

2.3. Greedy COPKM v2

Puesto que COPKM deshecha la solución de la iteración anterior y solo actualiza en base a los nuevos centroides, haciendo que en ocasiones la infeasibility aumente y el algoritmo cicle, se decide modificar el código para no desperdiciar los resultados anteriores.

En términos de programación, el único cambio es no comenzar con una solución vacía en cada iteración y calcular la infeasibility no hasta el índice actual sino con toda.

Algorithm 8: Algoritmo COPKM v2

```
Input: Conjunto de datos, lista de restricciones
solution = Solución aleatoria ;
indexes = Conjunto de índices (uno por cada dato) ;
centroids = Conjunto con los centroides de cada clúster ;
Barajar indexes ;
while solution cambie do
    for i in indexes do
        inf = infeasibility de la solución actual ;
        foreach cluster valido do
            newSol = solution modificando el cluster del índice i ;
            newInf = infeasibility de newSol ;
            if newInf < inf then
                Almacenar como cluster posible ;
            else if newInf < inf then
                Añadir a la lista de clusters posibles ;
        end
        distances = Distancias de los elementos a cada cluster ;
        solution se modifica con el cluster de los posibles con menor distancia ;
    end
end
return solution, centroids
```

3. Experimentación

Semillas	
Ejecución 1	949004259
Ejecución 2	589741062
Ejecución 3	277451237
Ejecución 4	49258669
Ejecución 5	3773969821

	Iris				Ecoli				Rand			
	TasaC	TasaInf	Agr	T	TasaC	TasaInf	Agr	T	TasaC	TasaInf	Agr	T
BL	0.92	141.20	1.82	27.50	46.74	1212.80	79.28	733.33	1.12	106.00	1.88	29.21
COPKM v1	0.67	3.60	0.56	2.24	37.04	201.40	42.44	432.70	0.77	6.20	0.81	1.60
COPKM v2	0.67	0.00	0.67	2.44	37.50	57.20	39.01	117.84	0.76	0.00	0.76	2.02

Cuadro 1: Resultados medios de todos los algoritmos para 10 % de restricciones

	Iris				Ecoli				Rand			
	TasaC	TasaInf	Agr	T	TasaC	TasaInf	Agr	T	TasaC	TasaInf	Agr	T
BL	0.90	264.80	1.74	40.18	46.85	2175.20	76.03	1242.45	1.15	293.20	2.21	38.25
COPKM v1	0.67	3.40	0.68	3.55	35.08	169.40	37.36	274.13	0.77	13.60	0.82	2.66
COPKM v2	0.67	0.00	0.67	3.69	31.10	0.00	31.10	145.25	0.76	0.00	0.76	3.26

Cuadro 2: Resultados medios de todos los algoritmos para 20 % de restricciones

3.1. Greedy COPKM v1

	Iris				Ecoli				Rand			
	TasaC	TasaInf	Agr	T	TasaC	TasaInf	Agr	T	TasaC	TasaInf	Agr	T
Ejecución 1	0.67	7.00	0.06	2.27	36.48	153.00	40.59	184.56	0.76	0.00	0.76	2.59
Ejecución 2	0.67	0.00	0.67	2.33	38.22	168.00	42.73	148.97	0.76	0.00	0.76	1.31
Ejecución 3	0.67	0.00	0.67	2.41	35.61	216.00	41.40	868.75	0.81	31.00	1.04	1.77
Ejecución 4	0.67	11.00	0.74	2.33	37.99	296.00	45.93	395.77	0.76	0.00	0.76	1.14
Ejecuión 5	0.67	0.00	0.67	1.88	36.90	174.00	41.57	565.45	0.76	0.00	0.76	1.19
Media	0.67	3.60	0.56	2.24	37.04	201.40	42.44	432.70	0.77	6.20	0.81	1.60

Cuadro 3: Resultados COPKM v1 para 10 % de restricciones

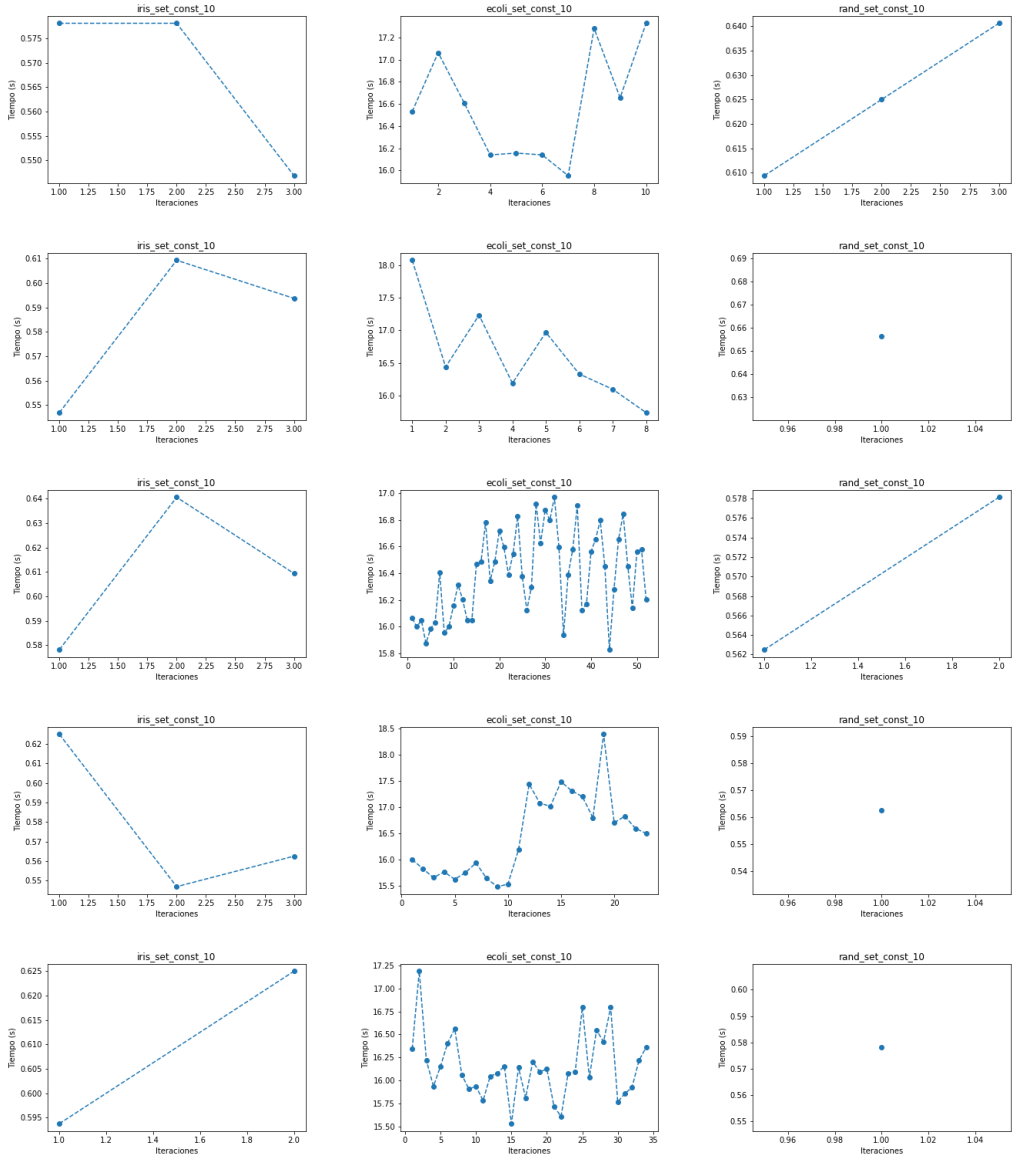

Figura 1: Tiempos de COPKM v1 para 10 % de restricciones

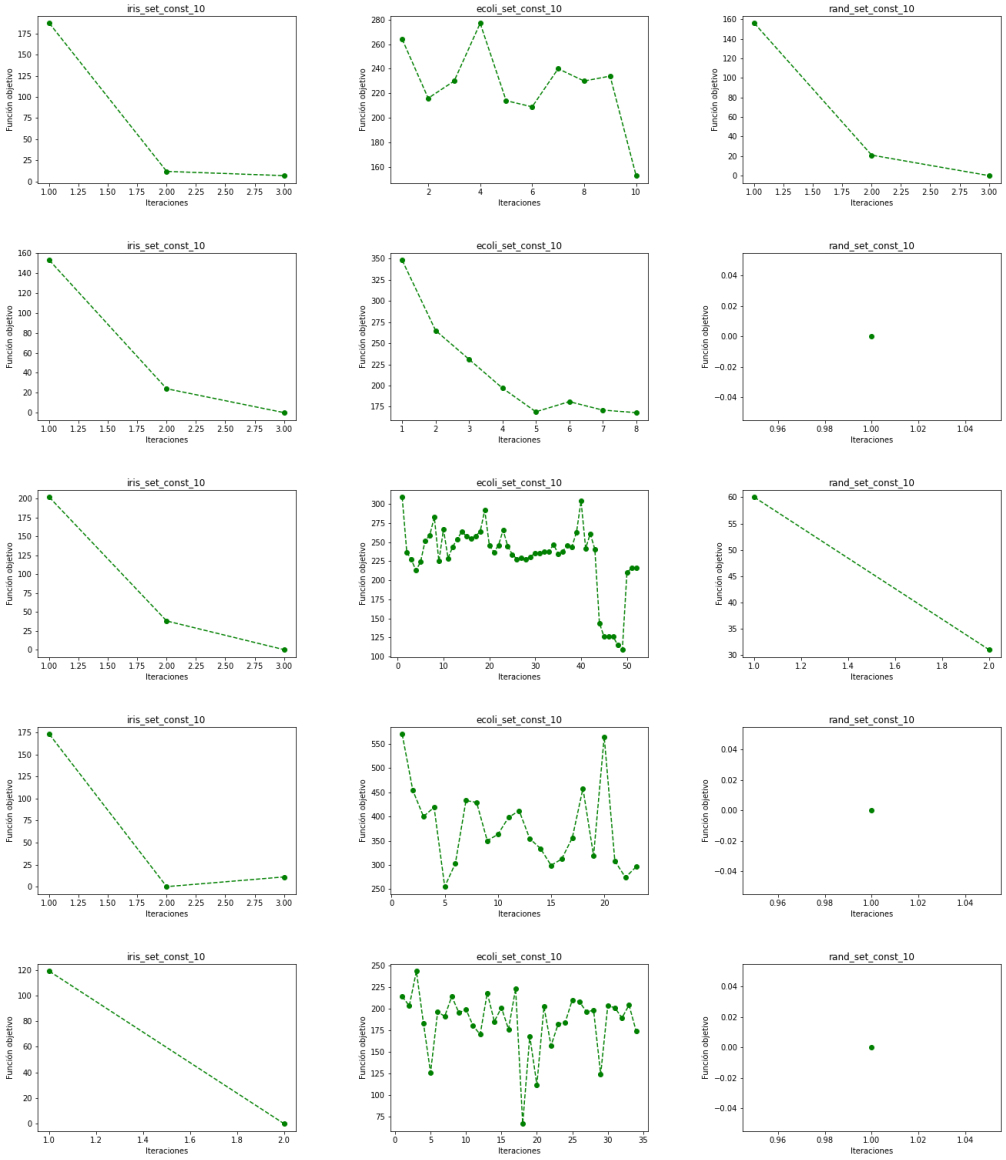

Figura 2: Agregado de COPKM v1 para 10 % de restricciones

	Iris				Ecoli				Rand			
	TasaC	TasaInf	Agr	T	TasaC	TasaInf	Agr	T	TasaC	TasaInf	Agr	T
Ejecución 1	0.67	17.00	0.72	3.36	34.44	351.00	39.15	169.77	0.76	0.00	0.76	3.38
Ejecución 2	0.67	0.00	0.67	2.33	37.64	158.00	39.76	283.50	0.76	0.00	0.76	2.19
Ejecución 3	0.67	0.00	0.67	5.56	33.53	128.00	35.25	366.92	0.83	68.00	1.08	3.28
Ejecución 4	0.67	0.00	0.67	3.22	32.35	92.00	33.58	392.08	0.76	0.00	0.76	2.31
Ejecuión 5	0.67	0.00	0.67	3.30	37.46	118.00	39.04	158.39	0.76	0.00	0.76	2.14
Media	0.67	3.40	0.68	3.55	35.08	169.40	37.36	274.13	0.77	13.60	0.82	2.66

Cuadro 4: Resultados COPKM v1 para 20 % de restricciones

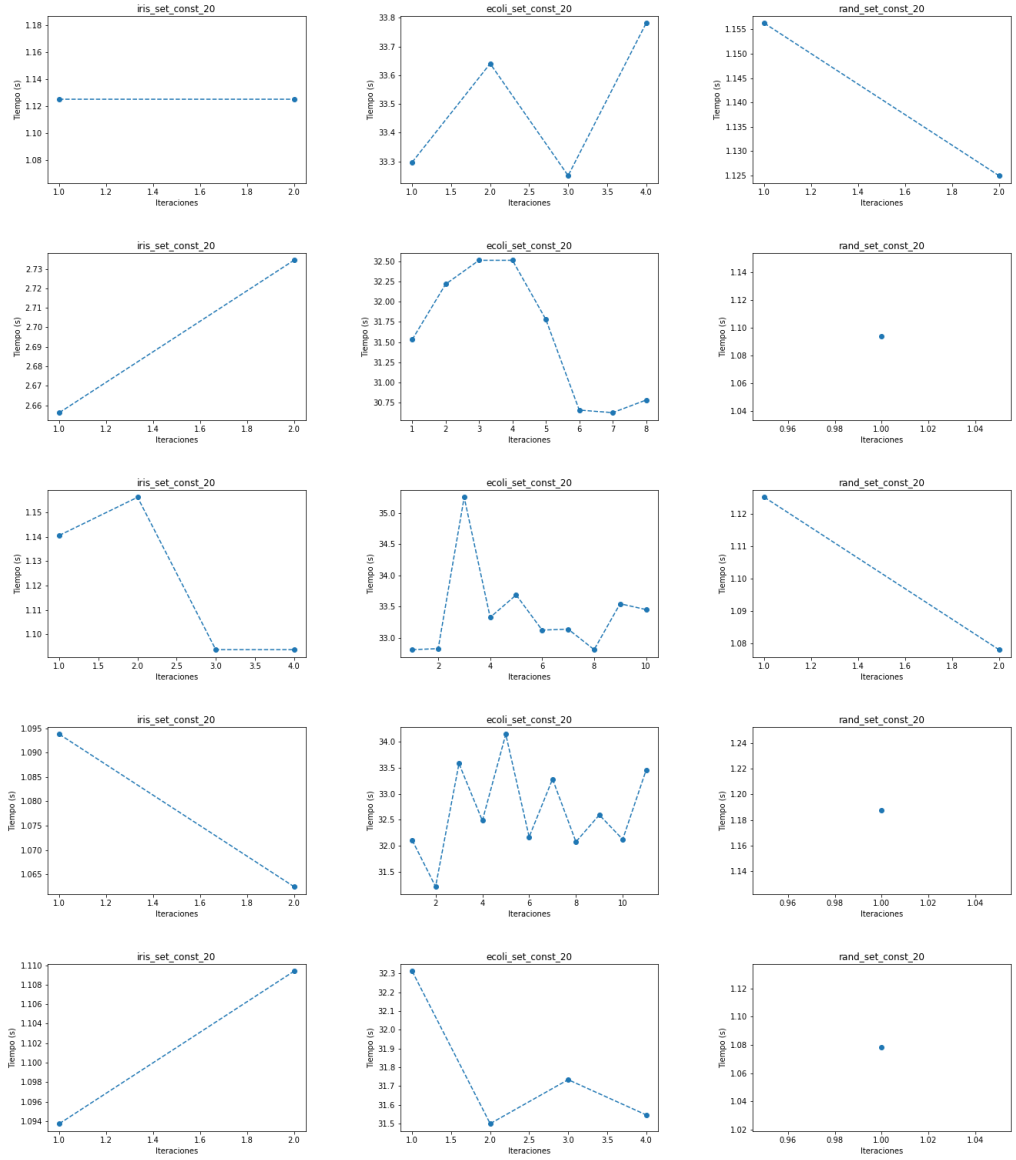

Figura 3: Tiempos de COPKM v1 para 20 % de restricciones

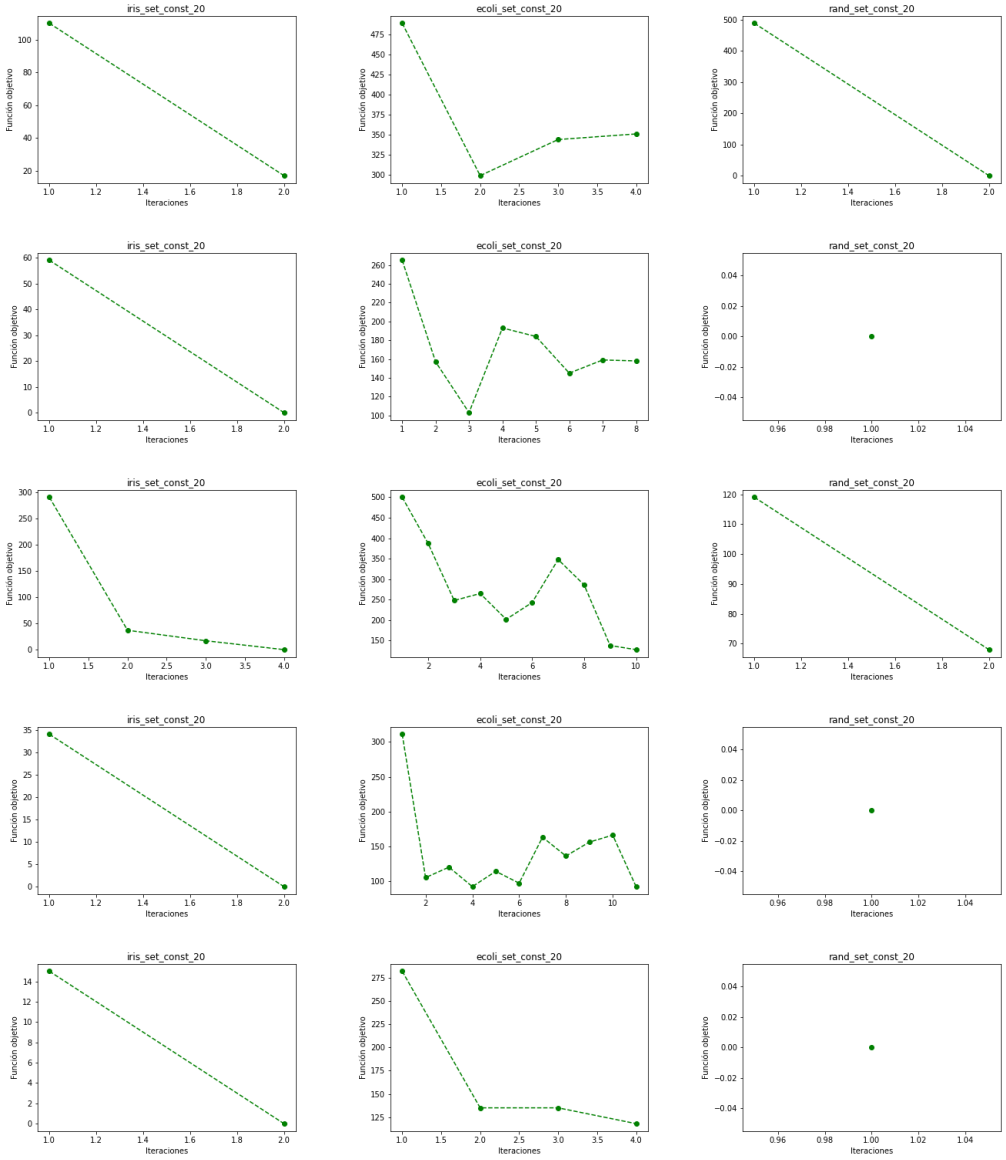

Figura 4: Agregado de COPKM v1 para 20 % de restricciones

3.2. Búsqueda Local

En todas las ejecuciones se utilizó un valor λ igual al cociente entre la mayor distancia entre dos elementos cualesquiera del conjunto de datos y el número de restricciones del problema.

$$\lambda = \frac{\text{máx } D}{|R|} \tag{5}$$

	Iris				Ecoli				Rand			
	TasaC	TasaInf	Agr	T	TasaC	TasaInf	Agr	T	TasaC	TasaInf	Agr	T
Ejecución 1	1.07	241.00	2.60	17.27	46.46	1188.00	78.34	747.25	1.22	131.00	2.17	27.70
Ejecución 2	1.02	193.00	2.24	20.77	46.63	1207.00	79.01	712.41	1.23	133.00	2.19	21.05
Ejecución 3	1.02	133.00	1.86	30.67	46.63	1228.00	79.58	741.86	1.14	133.00	2.10	35.20
Ejecución 4	0.76	75.00	1.23	29.66	47.39	1197.00	79.50	753.13	1.04	78.00	1.61	29.38
Ejecuión 5	0.76	64.00	1.16	39.13	46.60	1244.00	79.98	712.03	0.95	55.00	1.35	32.73
Media	0.92	141.20	1.82	27.50	46.74	1212.80	79.28	733.33	1.12	106.00	1.88	29.21

Cuadro 5: Resultados BL para 10 % de restricciones

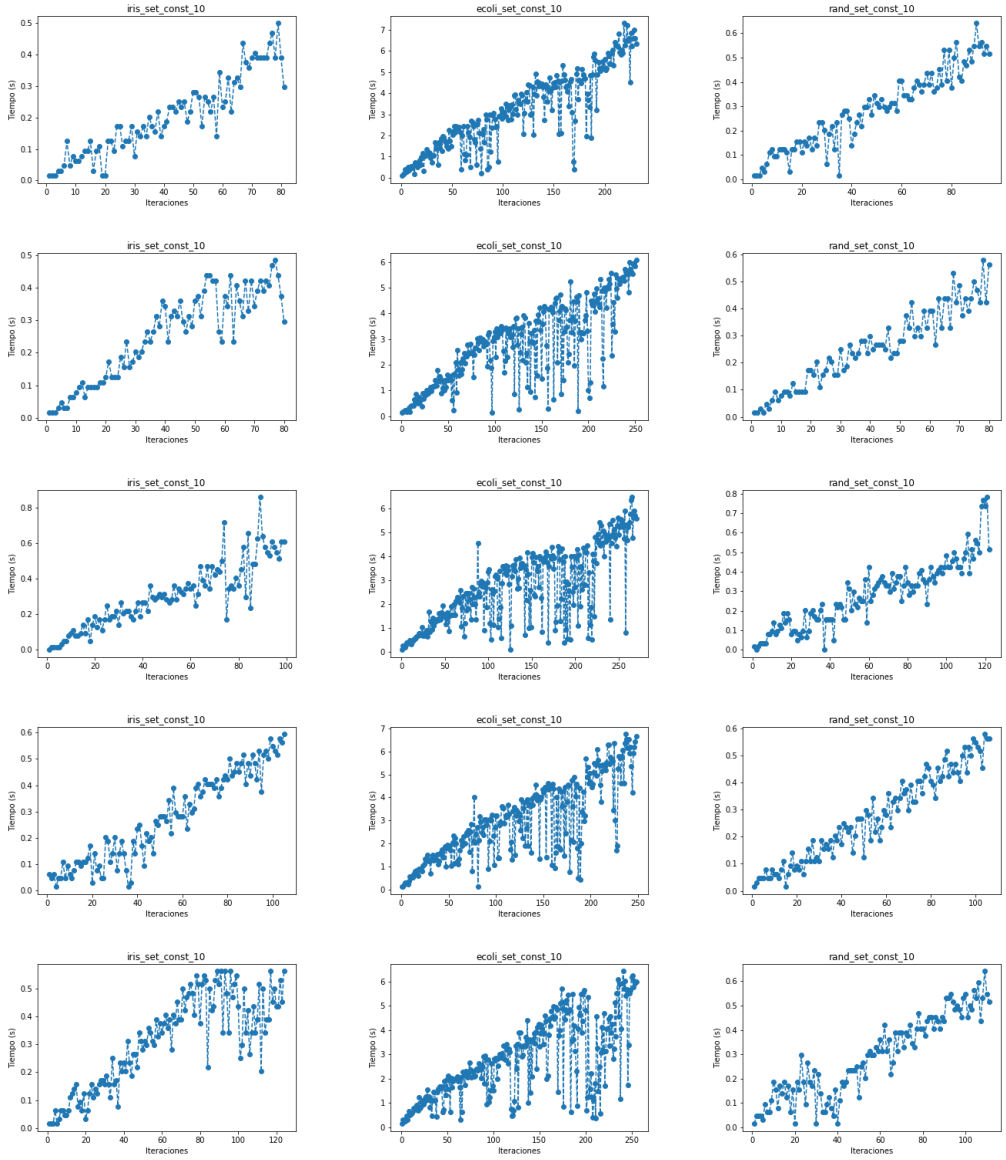

Figura 5: Tiempos de BL para 10 % de restricciones

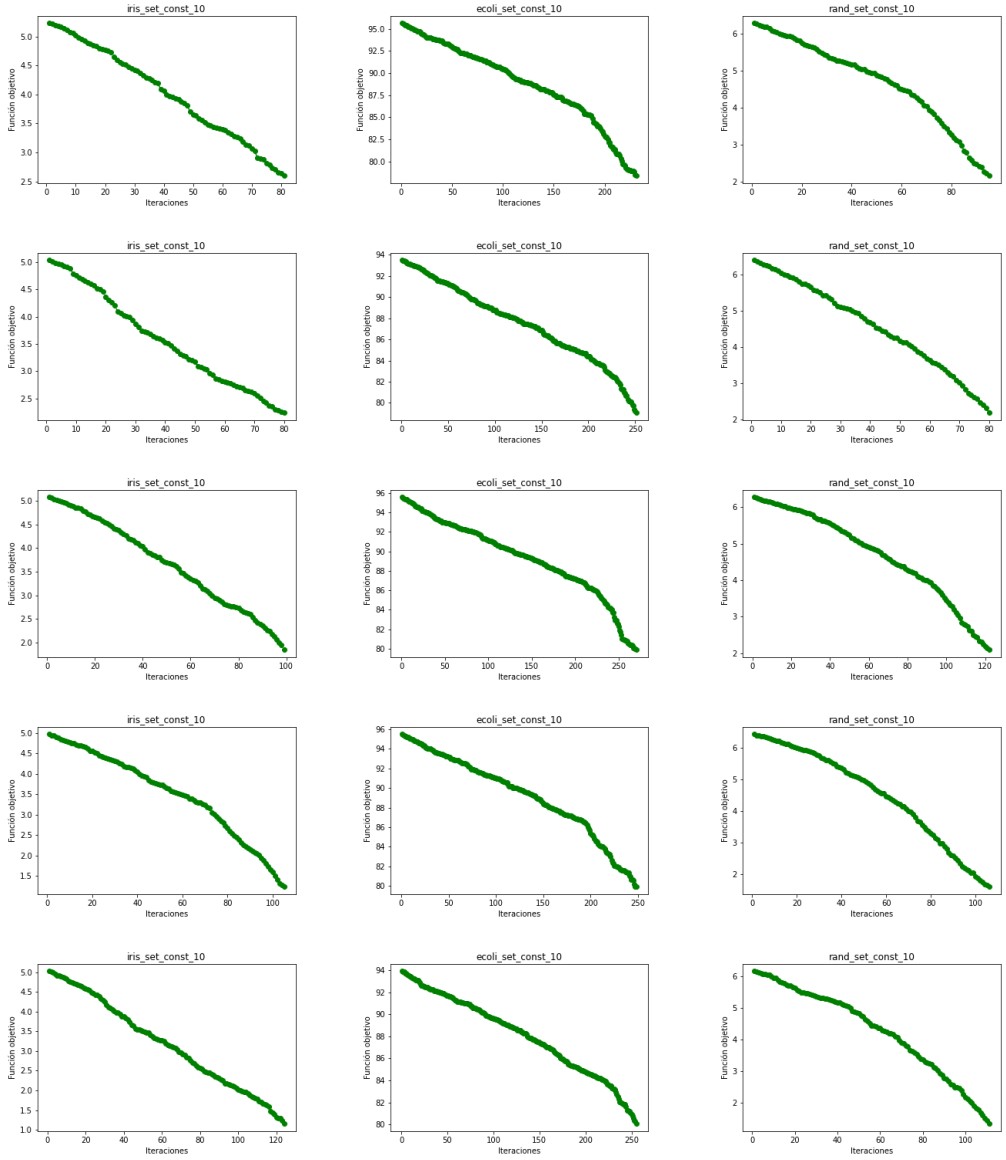

Figura 6: Agregado de BL para 10 % de restricciones

	Iris				Ecoli				Rand			
	TasaC	TasaInf	Agr	T	TasaC	TasaInf	Agr	T	TasaC	TasaInf	Agr	T
Ejecución 1	0.82	279.00	1.70	42.30	46.28	2133.00	74.89	1256.20	1.14	272.00	2.12	40.48
Ejecución 2	0.94	325.00	1.97	32.75	46.64	2194.00	76.07	1220.25	1.25	366.00	2.57	33.13
Ejecución 3	1.11	338.00	2.18	36.50	46.88	2178.00	76.10	1234.64	1.07	258.00	2.00	40.19
Ejecución 4	0.93	276.00	1.80	38.41	47.14	2190.00	76.52	1223.83	1.02	194.00	1.71	44.39
Ejecuión 5	0.72	106.00	1.05	50.94	47.30	2181.00	76.56	1277.31	1.30	376.00	2.65	33.05
Media	0.90	264.80	1.74	40.18	46.85	2175.20	76.03	1242.45	1.15	293.20	2.21	38.25

Cuadro 6: Resultados BL para 20 % de restricciones

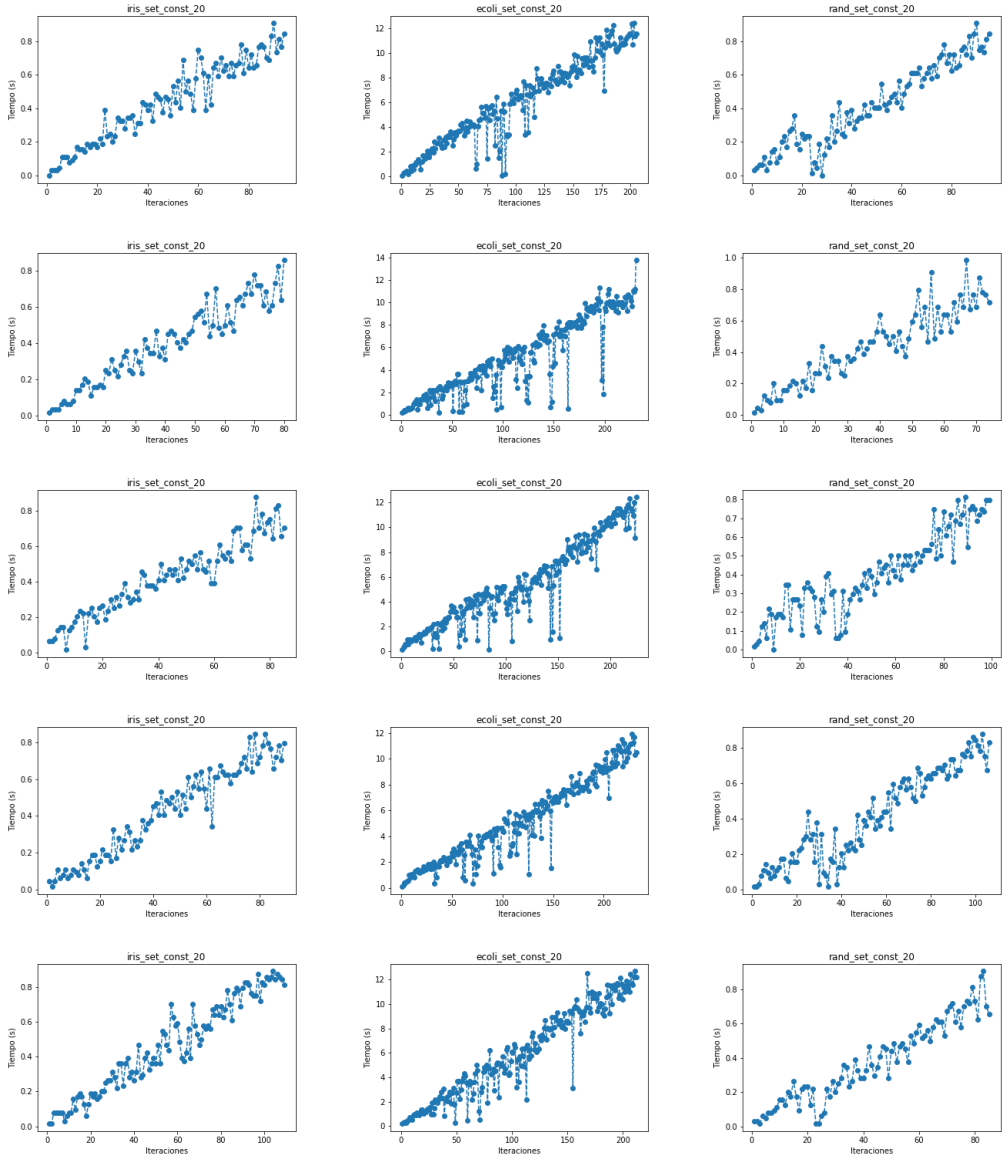

Figura 7: Tiempos de BL para 20 % de restricciones

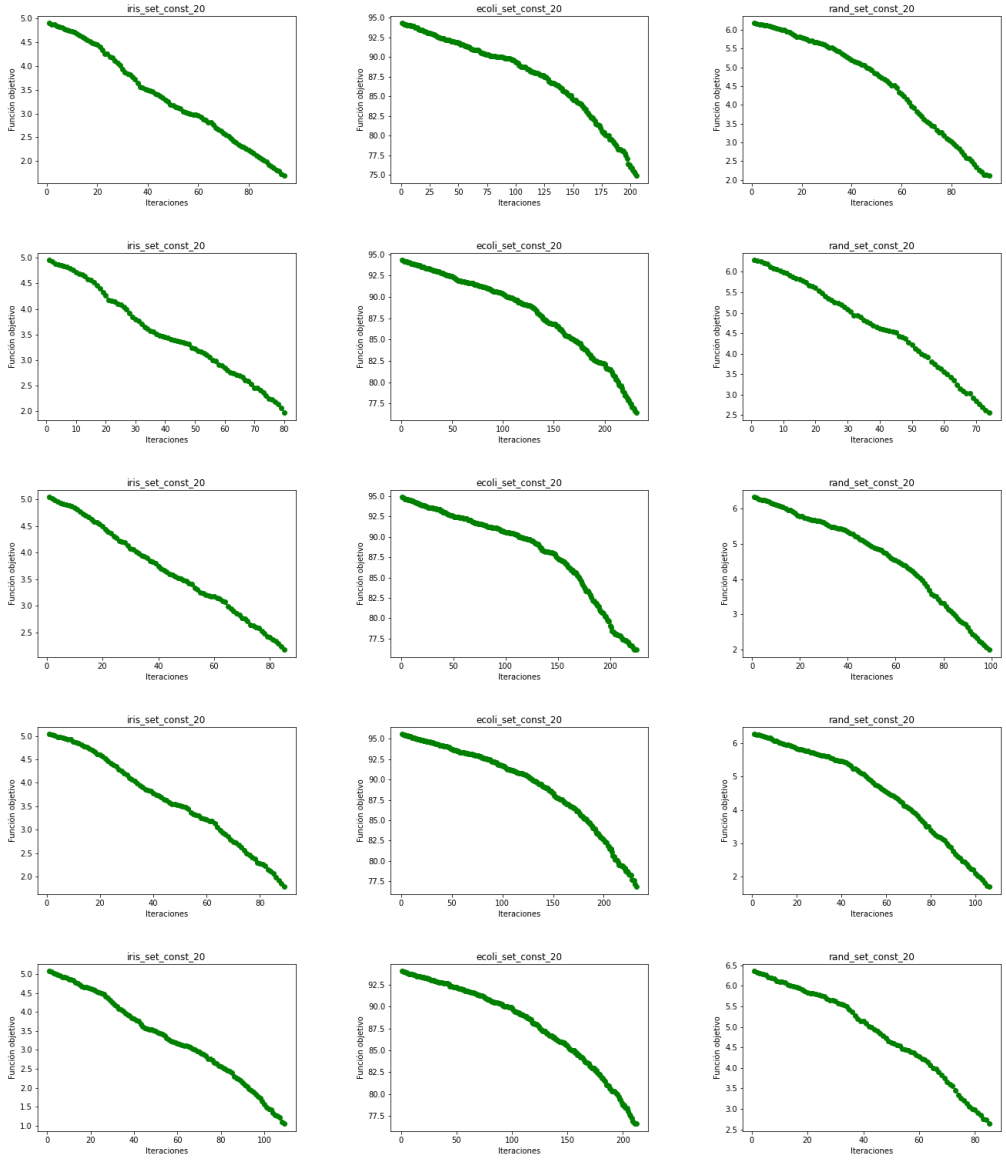

Figura 8: Agregado de BL para 20 % de restricciones

3.3. Greedy COPKM v2

	Iris				Ecoli				Rand			
	TasaC	TasaInf	Agr	T	TasaC	TasaInf	Agr	T	TasaC	TasaInf	Agr	T
Ejecución 1	0.67	0.00	0.67	2.11	34.74	2.00	34.80	89.83	0.76	0.00	0.76	2.25
Ejecución 2	0.67	0.00	0.67	2.72	29.37	0.00	29.37	152.09	0.76	0.00	0.76	1.66
Ejecución 3	0.67	0.00	0.67	2.89	57.53	277.00	64.96	59.25	0.76	0.00	0.76	2.27
Ejecución 4	0.67	0.00	0.67	2.22	33.71	4.00	33.71	106.02	0.76	0.00	0.76	1.69
Ejecuión 5	0.67	0.00	0.67	2.25	32.15	3.00	32.23	182.00	0.76	0.00	0.76	2.25
Media	0.67	0.00	0.67	2.44	37.50	57.20	39.01	117.84	0.76	0.00	0.76	2.02

Cuadro 7: Resultados de COPKM v2 para 10 % de restricciones

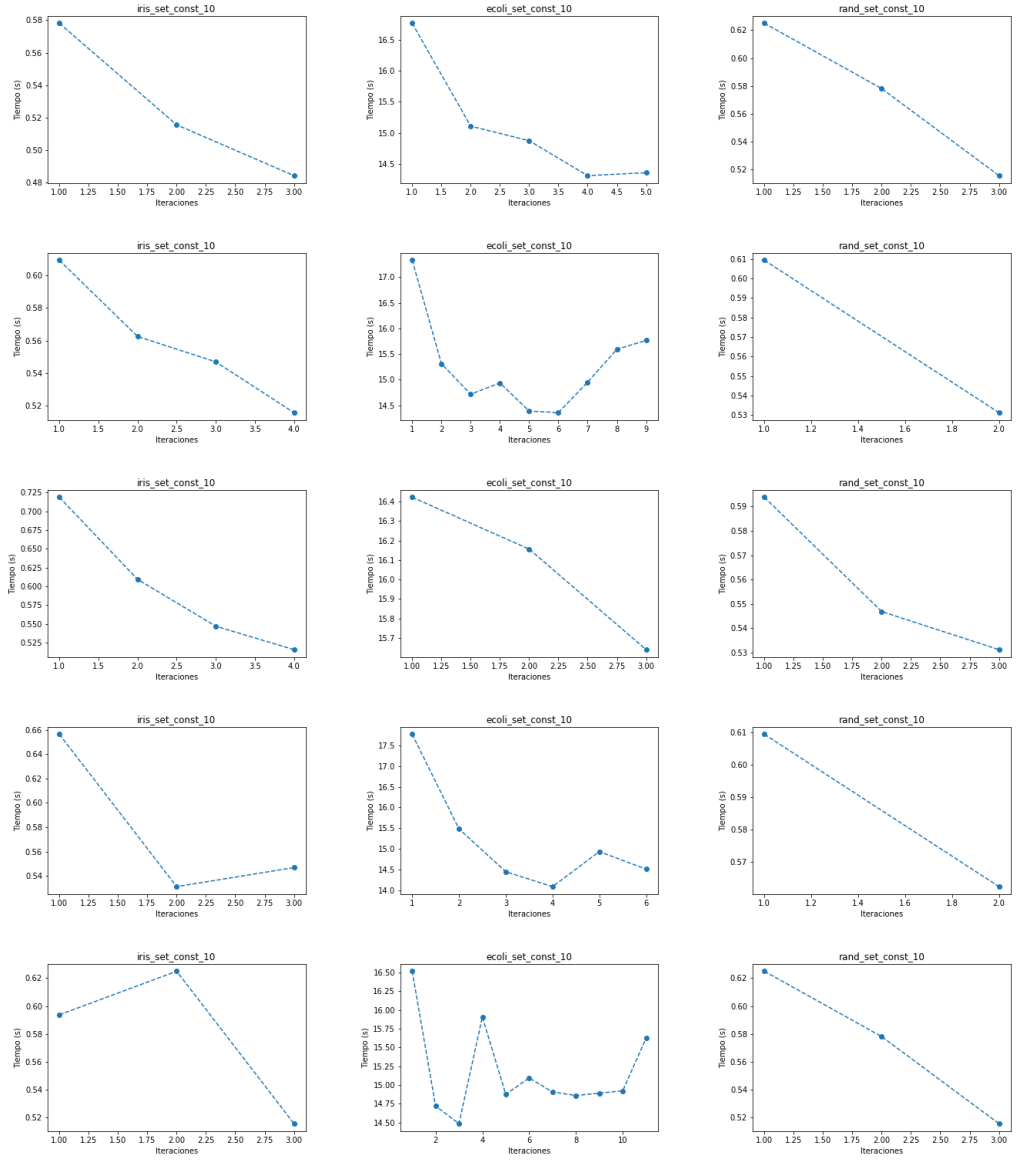

Figura 9: Tiempos de COPKM v2 para 10 % de restricciones

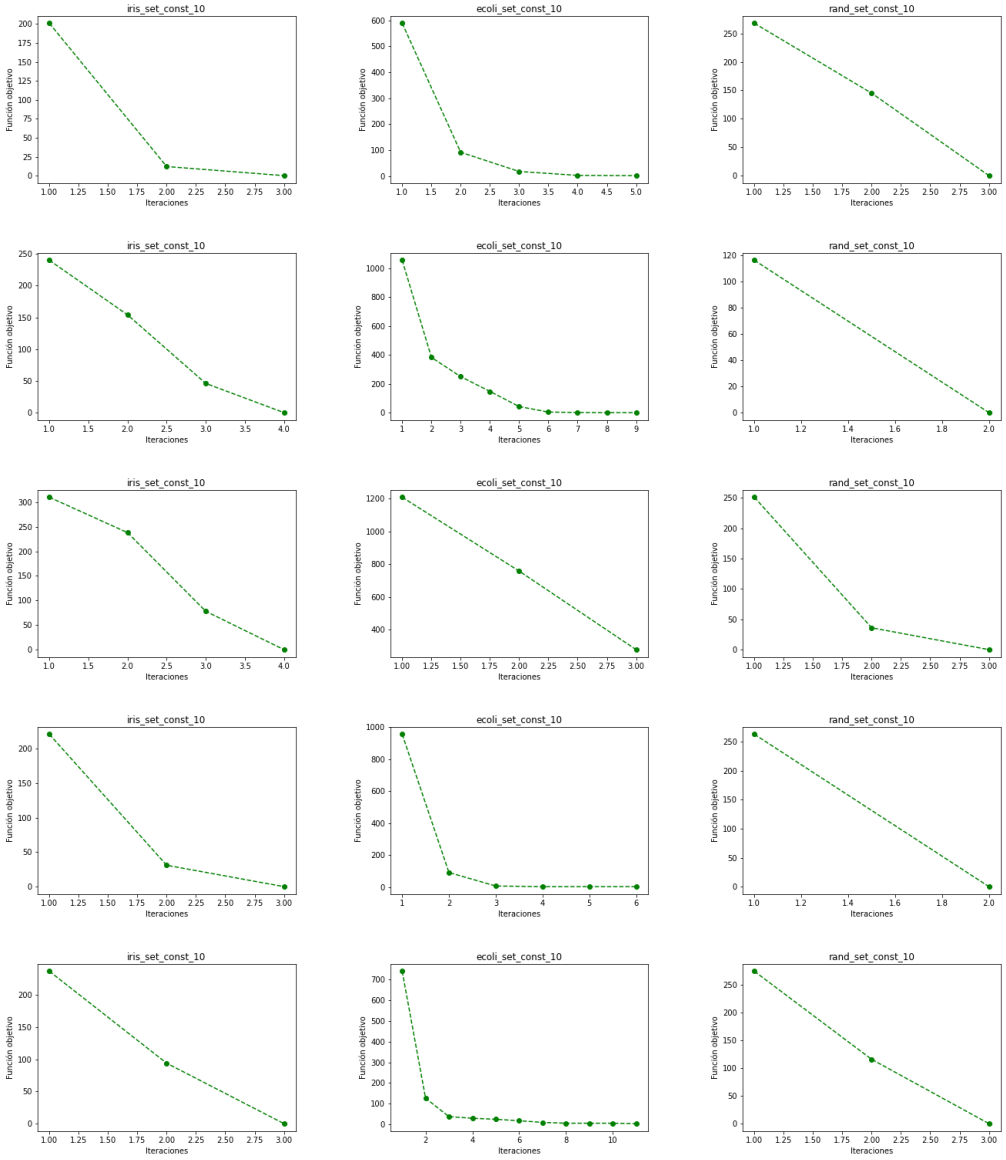

Figura 10: Agregado de COPKM v2 para 10 % de restricciones

	Iris				Ecoli				Rand			
	TasaC	TasaInf	Agr	T	TasaC	TasaInf	Agr	T	TasaC	TasaInf	Agr	T
Ejecución 1	0.67	0.00	0.67	3.25	29.61	0.00	29.61	183.00	0.76	0.00	0.76	3.19
Ejecución 2	0.67	0.00	0.67	4.42	33.46	0.00	33.46	149.48	0.76	0.00	0.76	3.34
Ejecución 3	0.67	0.00	0.67	4.33	34.43	0.00	34.43	93.83	0.76	0.00	0.76	3.30
Ejecución 4	0.67	0.00	0.67	3.17	29.61	0.00	29.61	149.45	0.76	0.00	0.76	3.34
Ejecuión 5	0.67	0.00	0.67	3.27	28.40	0.00	28.40	150.48	0.76	0.00	0.76	3.14
Media	0.67	0.00	0.67	3.69	31.10	0.00	31.10	145.25	0.76	0.00	0.76	3.26

Cuadro 8: Resultados de COPKM v2 para 20 % de restricciones

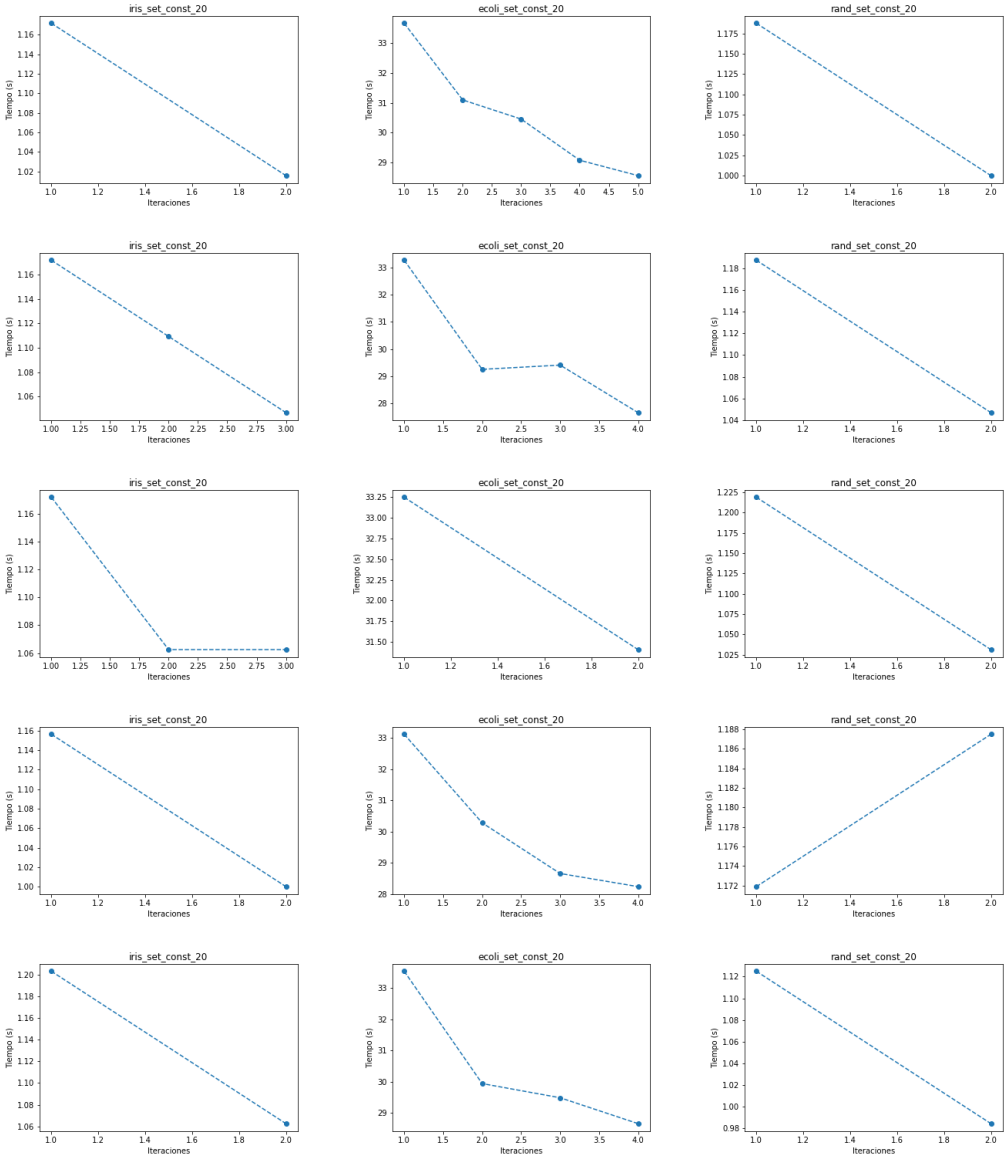

Figura 11: Tiempos de COPKM v2 para 20 % de restricciones

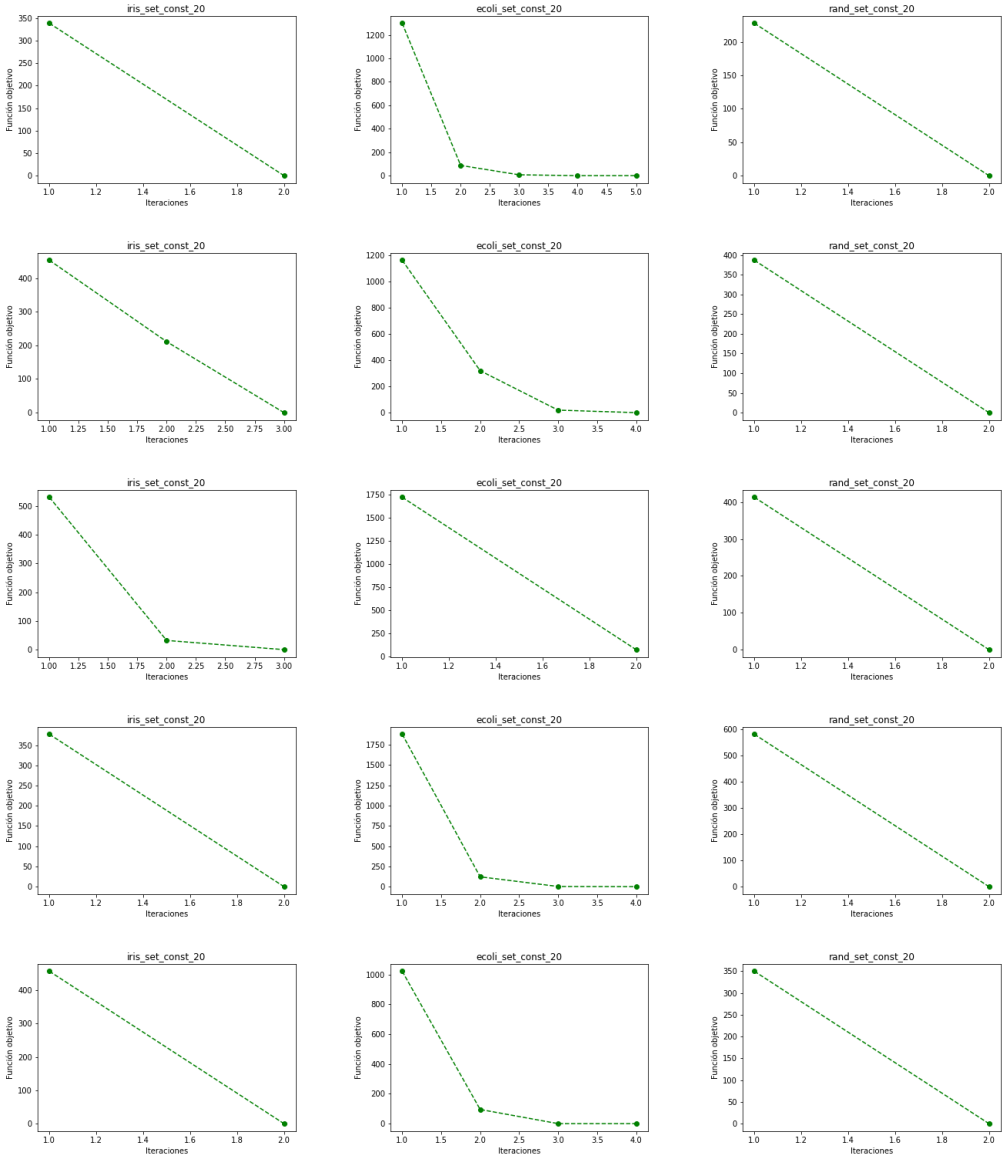

Figura 12: Agregado de COPKM v2 para 20 % de restricciones

4. Conclusiones

A partir de las soluciones obtenidas, y con la ayuda de las gráficas del Anexo A, nos damos cuenta que los conjuntos de datos Iris y Rand son extremádamente fáciles. No solo no tienen apenas superposición (en el caso de Rand ninguna), sino que no existe desordenación de los datos y por tanto soluciones del estilo [0..1..2..] son de muy buena calidad. Esto resulta en que las técnicas greedy, las cuales ordenan los clústers posibles a la hora de elegir uno de ellos, encuentren óptimos en una o dos iteraciones (dependiendo de la ordenación que haga en ese caso).

Seguidamente se realiza un análisis de los resultados de cada algoritmo, seguido de una comparativa entre ellos.

4.1. Greedy COPKM v1

Por un lado vemos que COPKM es un algoritmo bastante rápido y con muy buenas soluciones, donde salvo casos excepcionales encuentra la misma solución para Iris y Rand independientemente de la semilla. En estos conjuntos de datos, puesto que el algoritmo no tiene en cuenta las distancias intra-clúster, resulta obvio que aunque el valor de infeasibility sea cero la calidad de la solución no tiene por qué ser óptima, aunque viendo las gráficas del Anexo A parece que no se aleja mucho.

Además, vemos que la cantidad de iteraciones que hace en los problemas es corta en comparación con los de BL, esto se debe a que en cada iteración reduce la infeasibility de más de un elemento, mientras que en BL solo se cambia un único valor.

Por el comportamiento del algoritmo greedy vemos que se basa en mejorar la posición de los centroides para así fijar con mejor calidad los elementos a sus clústers. Pero la forma de hacerlo acarrea una desventaja, y es que no considera soluciones previas calculadas anteriormente. Esto hace que al seguir el mismo orden de índices en cada iteración los valores de infeasibility varían menos, puesto que hay que esperar a un elemento que tenga dos o más clústers posibles para poder actualizar en base al centroide.

4.2. Greedy COPKM v2

Añadiéndole la mejora al algoritmo greedy en COPKM v2, hacemos que se reutilice información de iteraciones anteriores, puesto que la infeasibility ya no se calcula únicamente con los elementos recién incluídos sino junto a los que ya estaban puestos (y a la espera de ser actualizados) en la iteración anterior.

Se podría decir que COPKM v2 actúa de manera similar a las mutaciones presentes en los algoritmos genéticos, pero sin permitir empeorar la infeasibility. Esto hace que la convergencia sea aún más rápida y, además, evitemos ciclos (puesto que si ordenamos los clústers posibles siempre cogerá los mismos en caso de empate).

Si nos fijamos en Ecoli, vemos que la calidad de las soluciones es mucho mejor, dónde especialmente supera a COPKM v1 es en tiempos de cómputo. El actualizar centroides y la solución en cada iteración facilita la decisión por una solución y la convergencia hacia un óptimo.

Al igual que en COPKM v1, vemos que los algoritmos greedy se comportan de mejor manera cuanto más aumenta el número de restricciones, esto es porque la selección de un clúster afecta más al valor de infeasibility.

4.3. Búsqueda Local

Observando los resultados de BL vemos que estos son bastante peores a los de los COPKMs. Esto se debe en teoría a dos motivos, la inicialización y al método de búsqueda en el espacio de soluciones.

Respecto al primero, BL busca converger hacia un mínimo a partir de una solución inicial. Esto hace que la selección de esta inicialización sea relevante en la calidad de la solución final, puesto que resulta muy fácil converger hacia mínimos locales. Algoritmos mixtos que sean capaces de empeorar su solución en ciertos momentos pueden ayudar a salir de estos focos locales y dirigirse hacia mejores resultados. Puesto que en el algoritmo COPKM v1 no consideramos soluciones previas, resulta claro que esto pase y pueda encontrar soluciones de mejor calidad.

Por otro lado, la exploración del espacio en BL tiene dos detalles. La selección de primero el mejor reduce el tiempo de cómputo pero también puede ocasionar el problema mencionado en el párrafo anterior. Un posible vecino puede desviar el resultado a un mínimo local de peor calidad.

Además, sabemos que la búsqueda local no se centra únicamente en la mejora de infeasibility, sino también en crear una solución con unos clústers bien diferenciados. Esto vienen dado por el valor λ, que en nuestro caso se ha inicializado a un valor pequeño. Las consecuencias de esta inicialización se aprecia en los valores de tasaC e infeasibility. Mientras que los greedy bajan mucho la infeasibility BL ataca en la tasaC para reducir el agregado (en greedy los valores de tasaC siguen siendo bajos en comparación con BL, pero esto es una consecuencia indirecta de los conjuntos de datos).

Adicionalmente, aunque en el peor de los casos todos los algoritmos exploran el mismo tamaño de vecindario, los greedy ganan en velocidad en gran medida. Esto está causado por el tiempo perdido en la generación y evaluación de un nuevo vecino, además de la necesidad de comprobar las distancias para cada uno de ellos.

Puesto que en BL el vecindario son todos los posibles cambios, a medida que mejora la solución debemos explorarlo más para encontrar uno que mejore (se aprecia en las gráficas de tiempo, donde para los greedy son más inestables pero crecen menos). Mientras exista, el tiempo entre iteraciones aumenta drásticamente para BL. Pero puesto que tratamos con números reales no podemos comparar directamente con $f'(s) < f(s)$. Es necesario que este $<$ lo sea por un margen concreto. Este margen también es influyente en la solución y, aunque por la definición del algoritmo debería ser lo más bajo posible, un valor más pequeño hace que sea más sencillo saltar hacia un vecino y alarga el tiempo de cómputo.

Como se ha mencionado anteriormente, aunque por la forma de funcionar BL se piensa que es muy dependiente de la solución inicial, pues solo modificamos un elemento de la solución en cada iteración y no elegimos empeoramientos en ningún caso, en Ecoli no se da esta situación. Los valores de agregado acaban resultando muy similares independientemente de la semilla que se utilice. Esto se debe probablemente al pequeño número de evaluaciones en Ecoli. De esto último nos damos cuenta viendo las gráficas del valor de agregado por iteración, que nos indica que BL agota el número de evaluaciones permitido cuando aún podría seguir convergiendo.

Para Iris y Rand si se aprecia diferencia, más en el valor de infeasibility que en el de la tasaC.

A. Representación visual de primeras dimensiones

A.1. Greedy COPKM

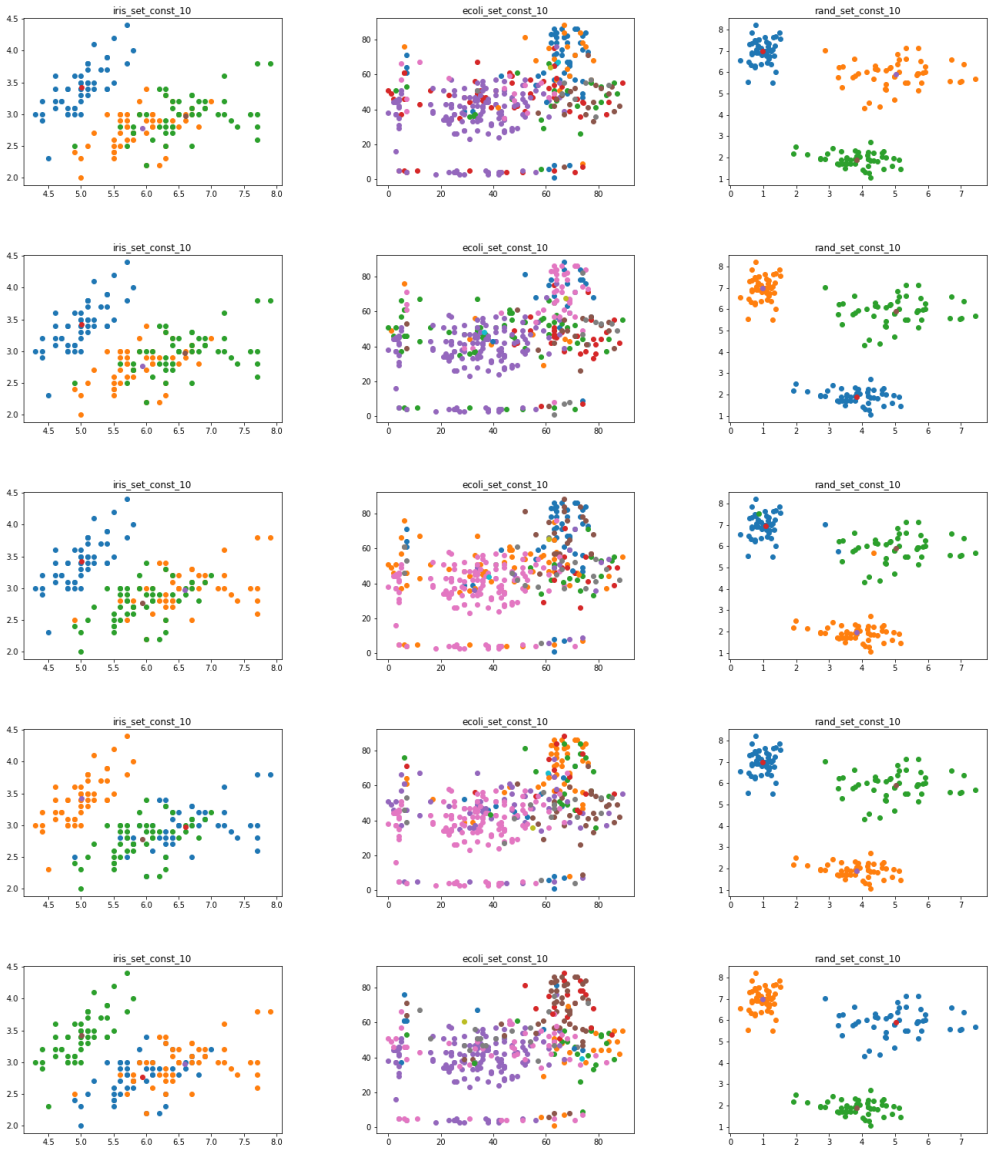

Figura 13: Primeras dimensiones de COPKM para 10 % de restricciones

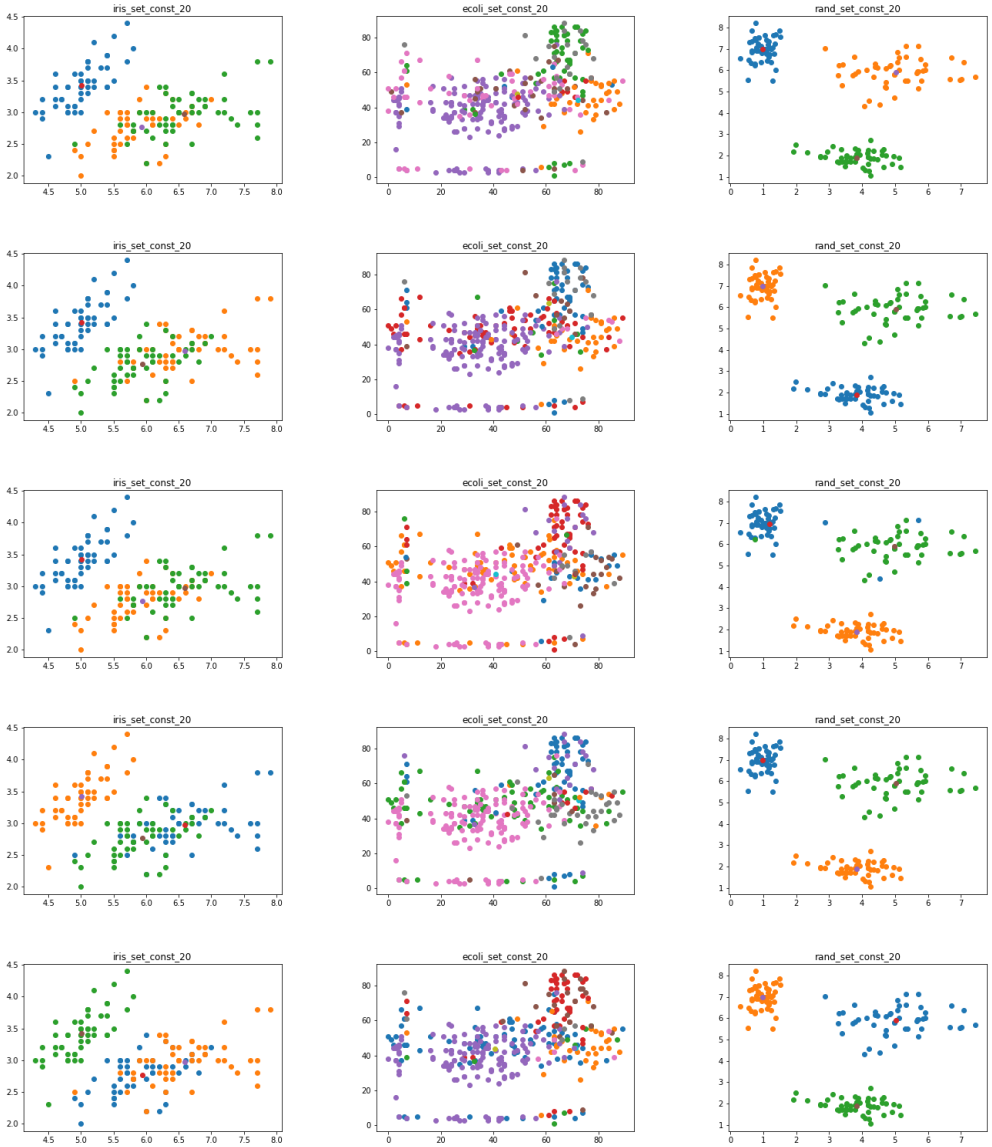

Figura 14: Primeras dimensiones de COPKM para 20 % de restricciones

A.2. Búsqueda Local

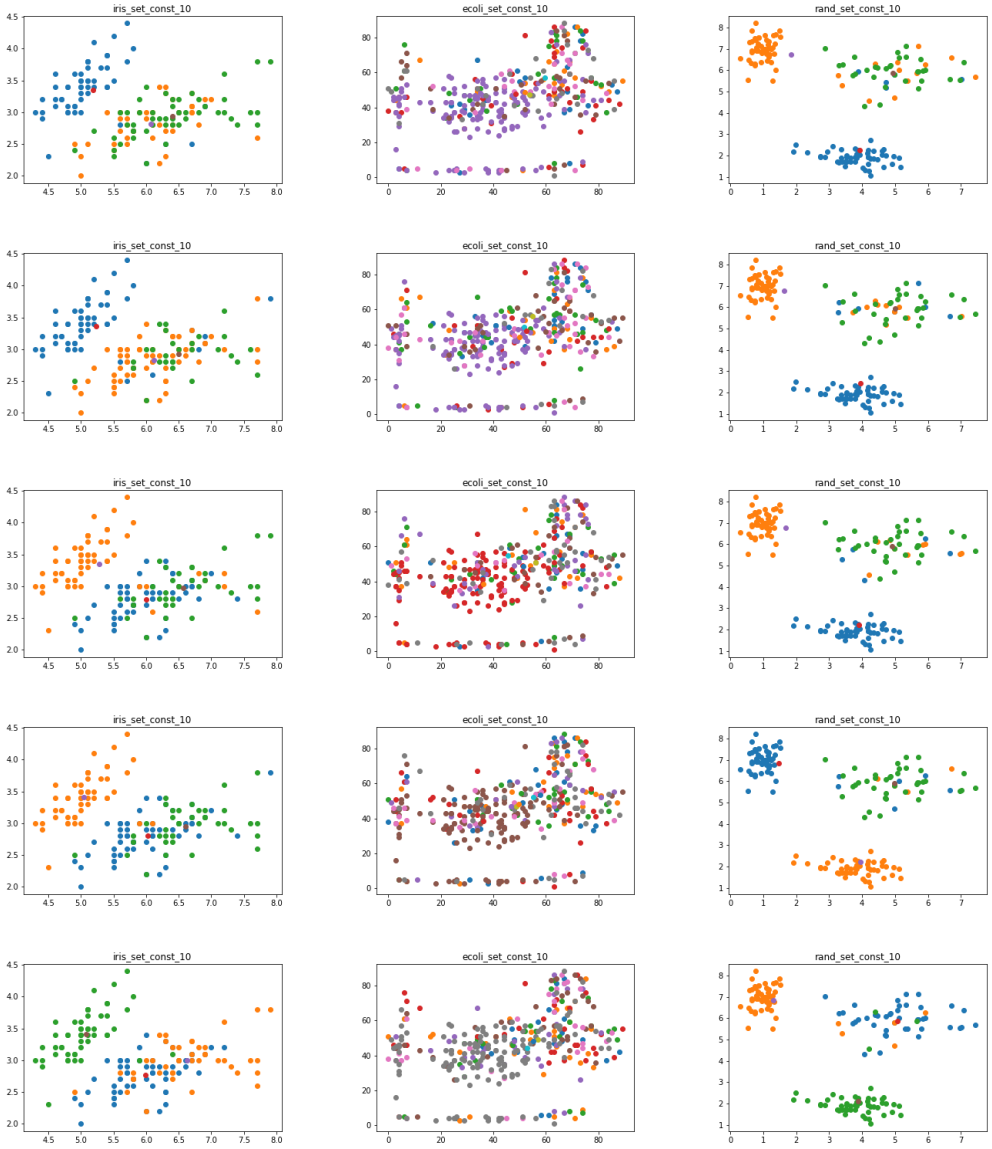

Figura 15: Primeras dimensiones de BL para 10 % de restricciones

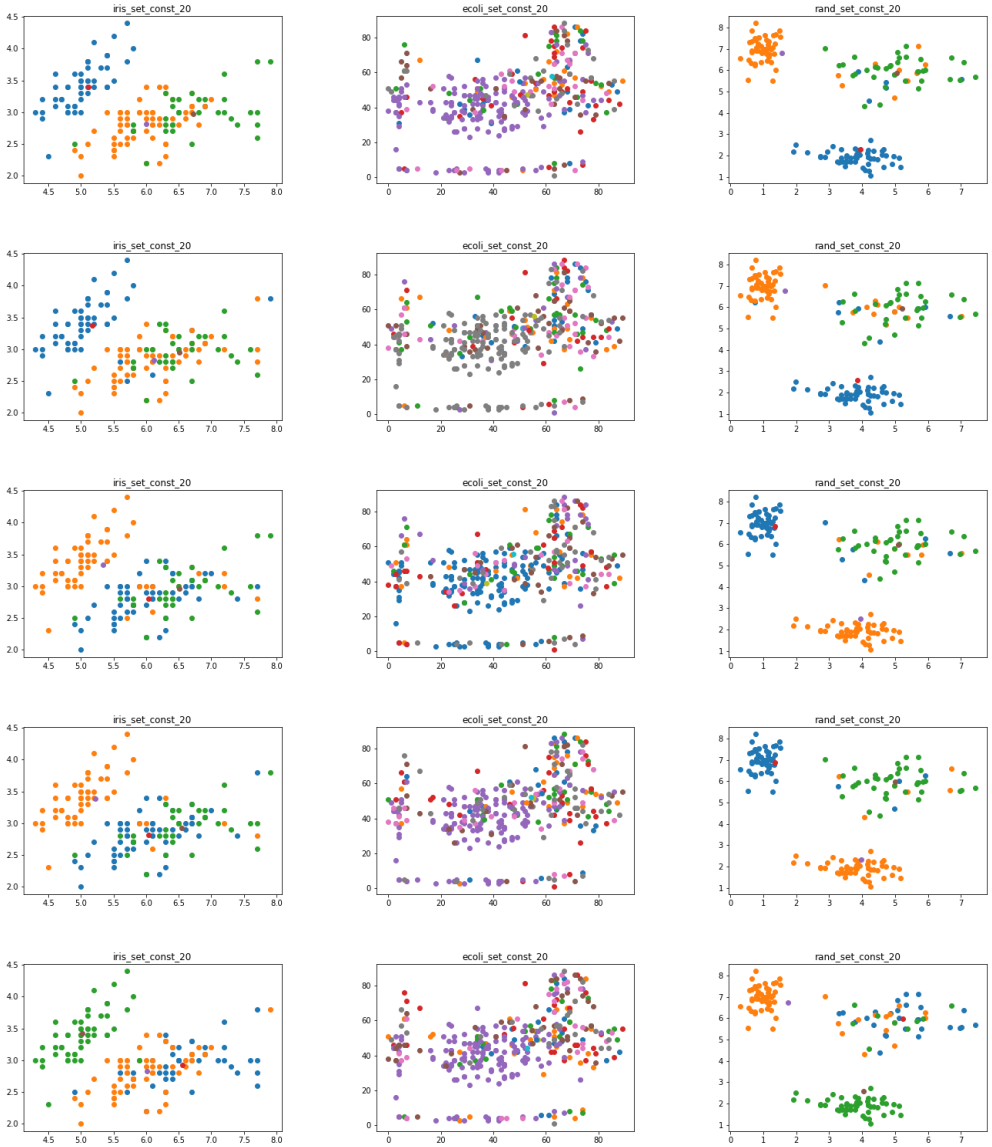

Figura 16: Primeras dimensiones de BL para 20 % de restricciones

A.3. Greedy COPKM v2

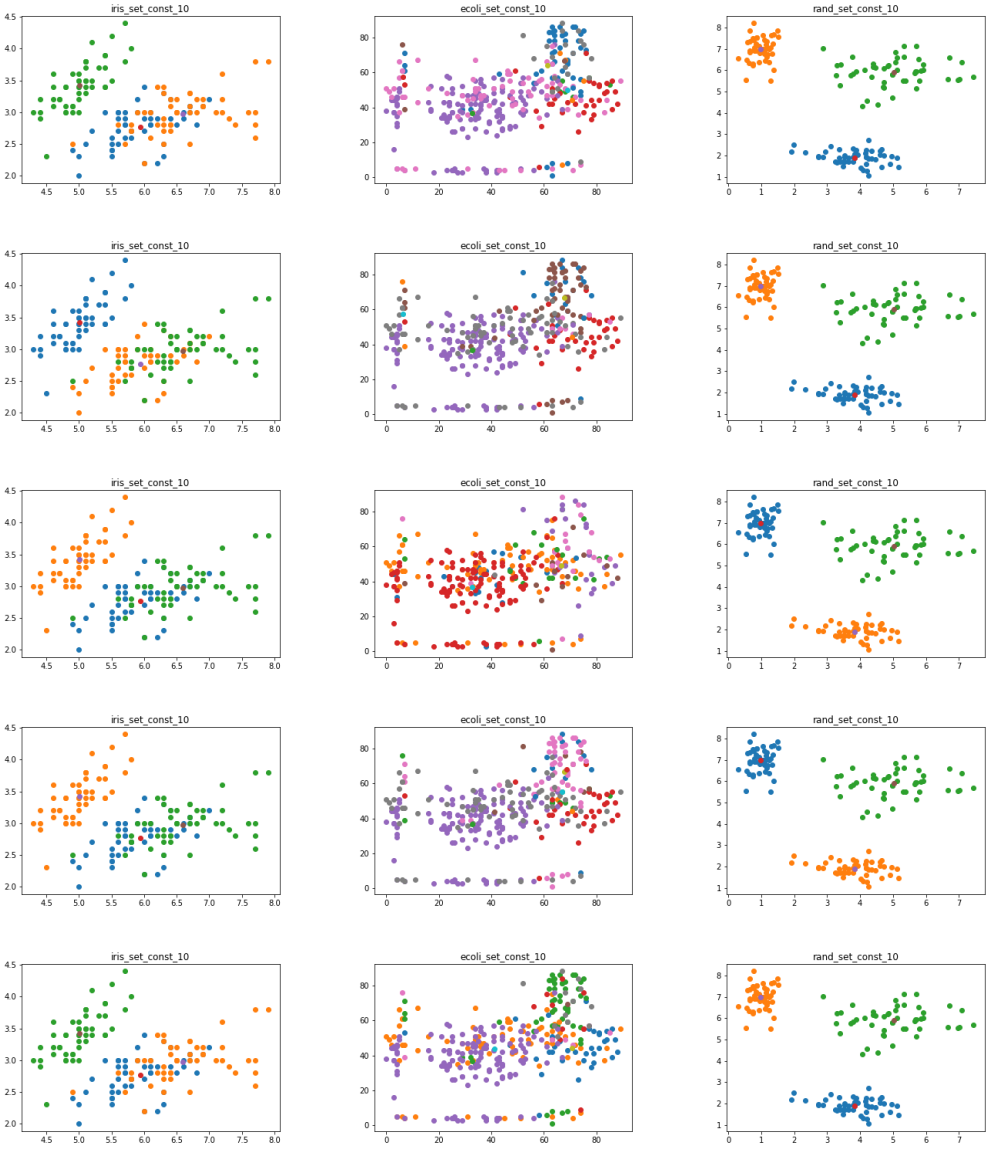

Figura 17: Primeras dimensiones de COPKM v2 para 10 % de restricciones

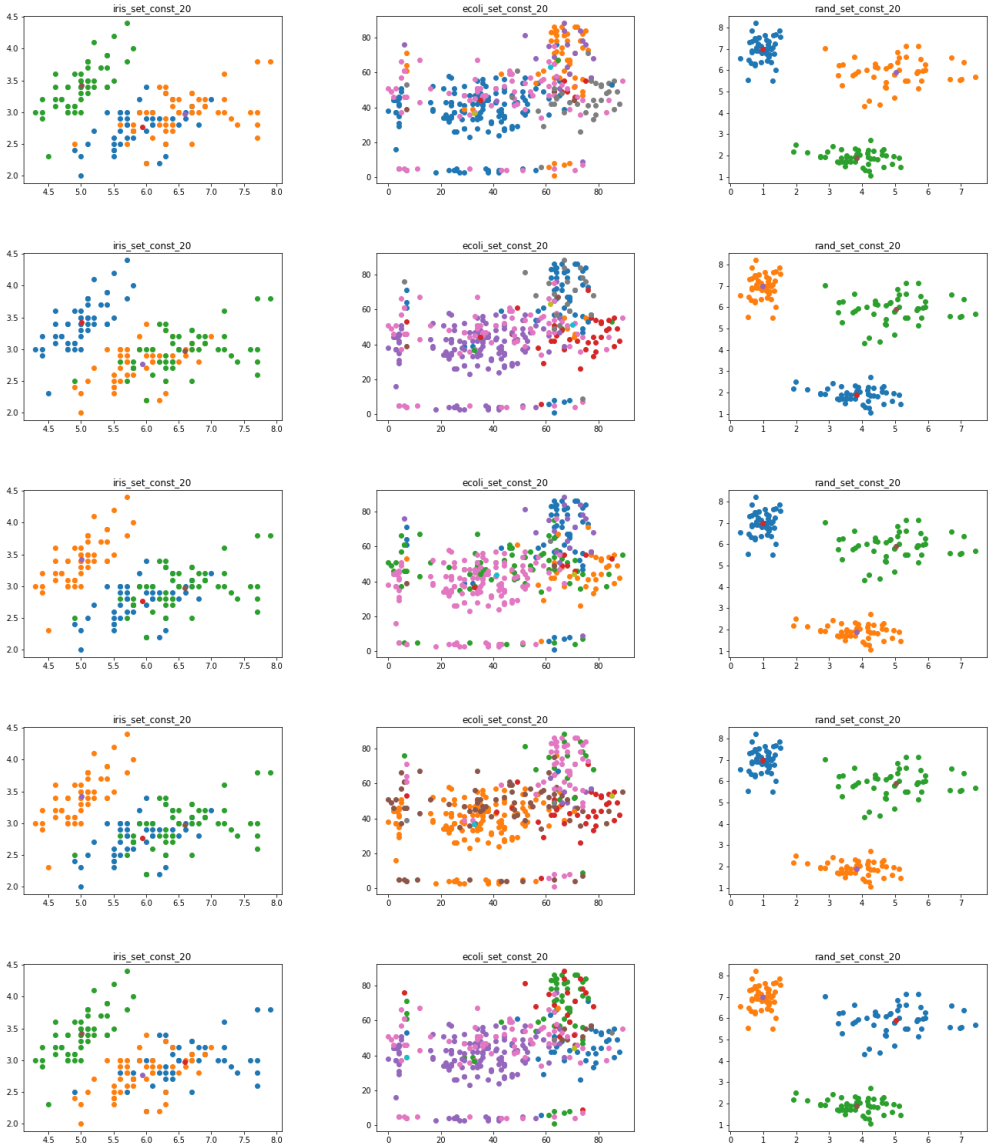

Figura 18: Primeras dimensiones de COPKM v2 para 20 % de restricciones

Referencias

[1] Numpy. `https://numpy.org/`.

[2] Scikit-learn. `https://scikit-learn.org/stable/`.