UNIVERSIDAD DE GRANADA

Metaheurísticas

Proyecto Final

Problema: Agrupamiento con Restricciones

Curso: 2019-2020
Autora: Alba Casillas Rodríguez
76738108B
3ºA – Grupo de Prácticas 1 (Miércoles 17:30 – 19:30)
albacaro@correo.ugr.es

Índice

1.Descripción del problema

PROBLEMA DEL AGRUPAMIENTO CON RESTRICCIONES

El agrupamiento, también llamado ***clustering***, es una tarea de aprendizaje no supervisado que consiste en agrupar un conjunto de objetos de tal manera que los objetos que estén en el mismo ***grupo (cluster)*** sean más ***similares*** entre sí que con los de otros grupos (clusters).

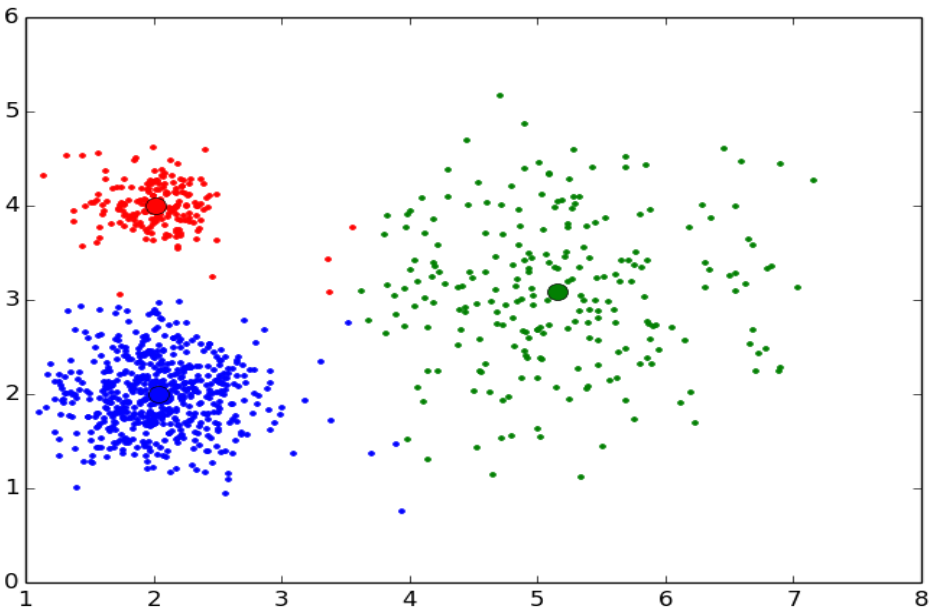
Esta similitud entre los objetos está definido mediante una ***distancia***, normalmente la Euclídea; por lo que haremos uso del término ***centroide***, siendo este el centro geométrico del cluster.

En nuestro problema, al clustering clásico le añadiremos restricciones; convirtiendo este problema en uno de aprendizaje semi-supervisado.

La partición de nuestro grupo de datos, deberá tener en cuenta ***restricciones fuertes***, que obligan a que todos los clusters deben contener al menos una instancia, cada instancia debe pertenecer a un único cluster, y la unión de los clusters debe ser el conjunto de datos.

Además, también deberán cumplirse unas ***restricciones débiles***, donde la partición del conjunto de datos debe minimizar el número de restricciones incumplidas (pudiendo incumplir algunas). Estas restricciones consisten en que dada una pareja de instancias, algunas deberán pertenecer al mismo cluster (***must-link***); y otras, por el contrario, no podrán estar en un mismo cluster (***cannot-link***).

El clustering es una tarea muy importante en exploración de data mining, y una técnica común en el análisis estadístico de datos, aprendizaje automático, reconocimiento de imágenes, etc.



2. Consideraciones del problema PAR

2.1- Representación de la solución :

Representamos nuestra solución como un vector de tamaño “n”, cuyos valores van desde 1 hasta “k”, siendo este el número de clusters definido.

$$\text{Solución} = \{S_1 , S_2 , \dots , S_n\} \setminus S_i \in K$$

Inicialmente, esta solución es creada mediante un generador de números aleatorios.

Deberemos asegurarnos de que nuestra solución sea **factible**, es decir, no dejaremos ningún cluster vacío.

2.2 – Función objetivo :

Será nuestra función a minimizar. Está se calculará mediante la fórmula:

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

- C será la desviación general.
- Infeasibility será el número de restricciones incumplidas.
- λ(lambda) será el parámetro de escalado para dar relevancia al infeasibility.

2.3 – Cálculo del Infeasibility:

Llamaremos infeasibility al número de restricciones incumplidas dada una pareja de instancias del conjunto de datos.

$$\text{Infeasibility} = \Sigma^{n}_{i=0} \Sigma^{n}_{j=i+1} V(x_i,x_j)$$

Se ha usado como estructura de datos una lista que almacena de manera:

[dato1, dato2, valor] , donde valor toma los valores: 1 (ML) o -1 (CL)

Por lo que el valor infeasibility se calcula:

```
def calcular_infeasibility(datos, v_solucion, restricciones):
    para i ϵ {0...len(restricciones)}:
        si  restricciones[i][2] == -1 && v_solucion[i][0] == v_solucion[i][1]:
            # INCUMPLE CANNOT-LINK
            infeasibility++
        si  restricciones[i][2] == 1 && v_solucion[i][0] != v_solucion[i][1]:
            # INCUMPLE CANNOT-LINK
            infeasibility++
```

2.4 – Generación de la población :

La población estará formada por “long_población” (de valor 50) indivudos.

Para ello, se ha creado una clase StellarObject compuesta por un constructor que inicializa a cero sus tres atributos : el vector solución, el valor de la función objetivo y los valores de los centroides. Esta clase solamente contendrá el constructor y una función usada como criterio de ordenación de los individuos (según qué tan bajo sea su valor de la función objetivo); no implementamos más métodos ya que simplemente queremos usar esta clase como una estructura equivalente a un “struct” que almacene los valores.

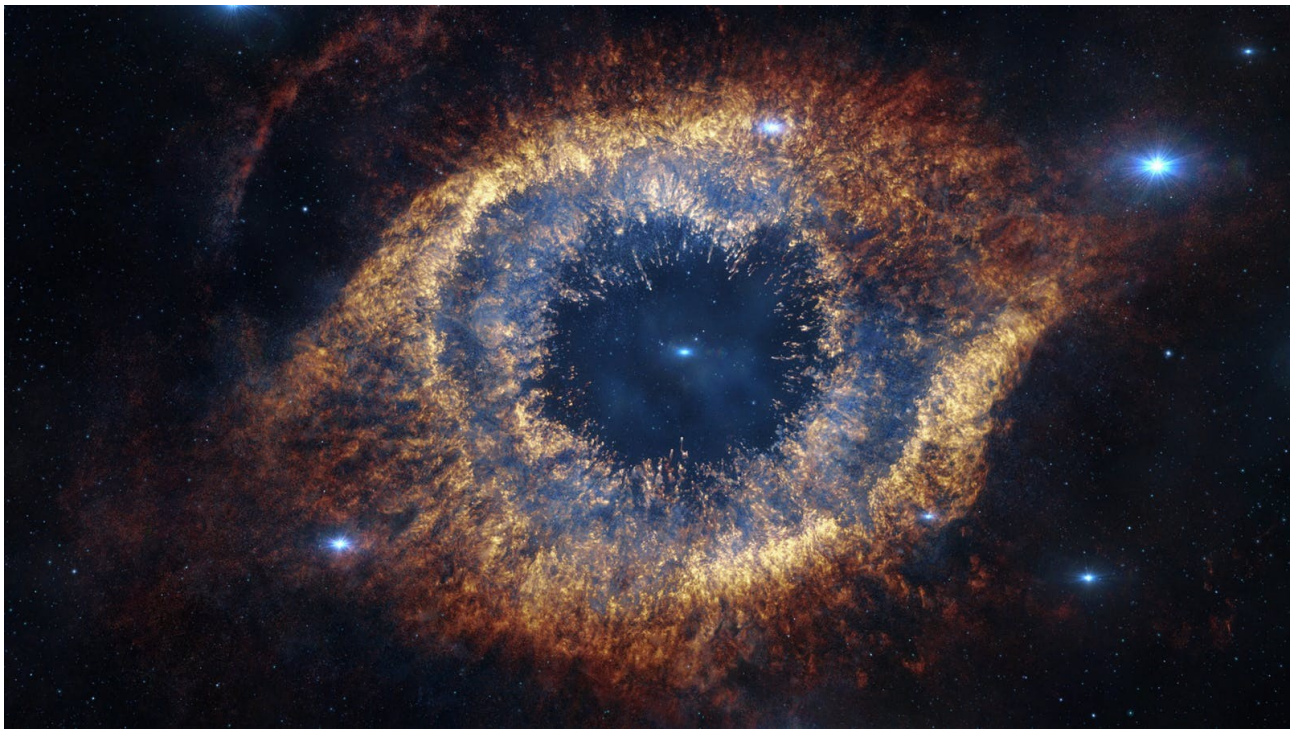
```
def generar_poblacion_aleatoria(long_poblacion, k, n_genes, d):
        poblacion = []

        for i ϵ {0...long_poblacion}:
                crom = StellarObject(n_genes,d) #Creamos nuestro objeto de la clase StellarObejct
                poblacion ← stellar
                for j ϵ {0...n_genes}:
                        poblacion[i].v_solucion[j] = random{1..k}

                for j ϵ{0...k}: #Las solcuiones tienen que ser factibles
                        si cluster_vacio(poblacion[i].v_solucion, j+1):
                                reparar( poblacion[i].v_solucion, j+1)

        return poblacion
```

3. Big Bang Big Crunch



Big Bang Big Crunch es una metaheurística derivada de la evolución de las teorías del universo en física y astronomía, describiendo como el universo fue creado, su evolución y como acabaría. Está compuesto por dos fases: Big Bang y Big Crunch.

El ***Big Bang (BB)*** representa un conjunto de procedimientos de disipación de energía en términos de aleatoriedad, lo que involucra la creación de una población inicial aleatoria de soluciones factibles (manteniendo cierta similitud con los algoritmos genéticos).

Las poblaciones producidas en el Big Bang se reducirán gradualmente en el Big Crunch. El ***Big Crunch (BC)*** es un procedimiento que distribuye partículas al azar y las re-ordena; es decir, manipula las soluciones hasta obtener una única solución de calidad. El objetivo de esto es reducir el tiempo de computo y producir una convergencia más rápida.

El coste de una solución representará la ***masa***. Por lo que la mejor solución (es decir, aquella que tenga un mejor coste) será el ***centro de la masa***. El centro de la masa atraerá soluciones potencialmente mejores, es decir, aquellas más cercanas al centro del espacio de búsqueda (universo) o el punto donde el Big Crunch convergerá.

La motivación con la que hemos elegido esta metaheurística se basa en el uso de una parametrización flexible del algoritmo, la cual promete una rápida convergencia a pesar de la existencia de múltiples mínimos locales. Además de la estrecha similitud con los algoritmos genéticos estudiados a lo largo de la asignatura, pudiendo ser esto un punto de discusión de los resultados interesante.

4. Adaptación de nuestro problema

Basándonos en el ***pseudocódigo general*** de la metaheurística elegida, la adaptaremos a nuestro problema de ***clustering***:

Big Bang (construcción de soluciones):
Paso 1: Generar una población de tamaño N (inicialmente las construiremos desde "el vacío", es decir, inicialmente serán soluciones factibles aleatorias ; en el resto de iteraciones serán generadas a partir del "conjunto de élite") y evaluar la población con nuestra función objetivo.

Big Crunch (Búsqueda en el paso de soluciones):
Repetir
Paso 2: **Generar** N_s vecinos para cada solución de la población y reemplazar al padre por el mejor descendiente para cada solución C_i de la población;
Paso 3: **Encontrar el centro de la masa** C_c;
Paso 4: Aplicar Búsqueda Local al centro de la masa;
Paso 5:; Actualizamos el conjunto de élite;
Paso 6: **Eliminamos las peores soluciones**;
Hasta que la población se reduzca a una única solución;
Paso 7: **Volver al paso 1 si no se ha cumplido del criterio de parada**;
Paso 8: **Devolver la mejor solución encontrada.**

En términos generales, BB-BC se asemeja a un Algoritmo Genético con Búsqueda Local; como en los criterios de evaluación del proyecto se evalúa por separado la metahurística y una mejora de esta añadiéndole Búsqueda Local u otra hibridación, de primeras **suprimiremos el Paso 4 del pseudocógido**, la cual lo añadiremos en la mejora.

Nuestro algoritmo comienza con la fase de construcción (Big Bang) donde construiremos para la primera generación una población de soluciones factibles aleatorias, la cual ya ha sido explicada en el apartado 2.4. de nuestro trabajo (Paso 1).

Para nuestro problema PAR, nos aseguramos de que nuestras soluciones sean factibles mediante una función auxiliar "reparar", la cual se encargará de revisar si hay algún cluster vacío. En el caso de haberlo, realizará una reparación en la que se le asignará a dicho cluster una posición aleatoria de

nuestra solución. Cada vez que hagamos una reparación, volveremos a comprobar si hay más clusters vacíos o si alguno se vació al reparar otro; para que la solución sea factible.

Para el resto de generaciones, la fase Big Bang se llevará a cabo generando soluciones a partir de modificaciones de las soluciones del conjunto de élite.

- *Conjunto de elite:* Este conjunto almacenará las mejores soluciones encontradas hasta el momento. El objetivo de esto es explotar la "memoria adaptativa", mediante una estructura de datos dedicada a almacenar y actualizar información relevante (en nuestro caso, una lista de las mejores soluciones obtenidas) para producir una búsqueda más efectiva y mejorar la producción de soluciones. Además, como se explicará más adelante, las soluciones de elite nos servirá de "límites" a la hora de la exploración de soluciones.

Como en el estudio hecho sobre los algoritmos genéticos se utilizaba una población de 50 individuos, he visto adecuado mantener el mismo tamaño de la población para este algoritmo. De esta forma, el tamaño del conjunto de élite serán los 10 mejores candidatos. Cuando generamos una nueva población a partir de este conjunto, se incluyen todas sus soluciones. El resto de individuos serán generados de manera que, para cada solución del conjunto de élite, se generarán varios vecinos mediante la técnica del cambio de clúster: elegiremos de forma aleatoria la posición de la solución a la que aplicarle el cambio y el nuevo valor de la misma; hasta generar completamente una nueva población. No hace falta añadir que en todo momento se comprobará la factibilidad de las mismas.

Tras esto, se evalúa y ordena (de menor a mayor coste) toda la población. En el B-BC original el coste viene dado en función de la distancia Euclídea , sin embargo, como adaptación a nuestro problema, se utilizará la fórmula usada a lo largo del curso, explicada en el apartado 2.2.

A continuación, pasamos a la fase de Big Crunch, donde primero generaremos vecinos para todas las soluciones de la población. En nuestro caso generamos 5 vecinos para cada solución mediante ciertas modificaciones. Cada padre será sustituido por el mejor de sus descendientes (es decir, aque que tenga menor valor objetivo) con el fin de conseguir soluciones de mayor calidad manteniendo la diversidad de la búsqueda e intentando no converger demasiado rápido (Paso 2).

En el BB-BC original, la generación de vecinos se rige por estas dos fórmulas:

$$C_i^{new} = C_c + \sigma \qquad \textit{(Ecuación 1)}$$

$$\sigma = \frac{r\alpha(C_{\max} - C_{\min})}{k}, 0 < \frac{\alpha}{k} < 1 \qquad \textit{(Ecuación 2)}$$

Donde C_i^{new} es el nuevo candidato de la solución C_i; y σ la desviación standard de una distribución normal.

Para el calculo de σ , "r" es un número aleatorio entre [0,1], "$\boldsymbol{\alpha}$" es una tasa de reducción cuyo valor también irá comprendido entre [0,1]. Los términos C_{max} y C_{min} serán los límites superior e inferior del conjunto de elite (es decir, el coste de la mejor y de la peor solución del conjunto), y "k" es el número máximo de evaluaciones.

En este punto es donde más adaptamos la metaheurística a nuestro problema. Como nuestro objetivo es modificar el vector solución de manera que haya una mejor distribución de los datos para disminuir el valor objetivo, hemos intentado darle un uso diferente a σ pero manteniendo la fórmula original.

En la implementación realizada, el centro de la masa C_c no representará el mejor coste obtenido, sino a la propia solución con mejor coste. Esto nos permitirá obtener diferentes vectores soluciones (los vecinos) pero nos imposibilita sumar el valor de σ a la solución.

Por ello, como σ es un valor numérico, he decidido que este sea el número de elementos a modificar del centro de la masa. Justifico esta decisión mediante varios motivos de peso:

- ***Motivo 1:*** Las posiciones del centro de la masa a modificar estimadas por σ serán sustituidas por valores del padre que está generando los vecinos; de esta manera, el descendiente tendrá mayor calidad al haber sido generado con la mejor solución, pero mantendrá una mayor diversidad manteniendo cierta correlación con su padre.

- ***Motivo 2*** : Como cada solución genera varios vecinos, la diferencia entre estos viene determinada por el parámetro "r", que al ser aleatorio hace que para cada descendiente se varíe un número distintos de posiciones. Además, el inicio del segmento a modificar también será elegido de manera aleatoria.

- ***Motivo 3 :*** Una de las ventajas de esta decisión será que mantendremos soluciones de calidad sin perder la esencia del BB-BC y manteniendo una relación estrecha entre el uso original de σ y el nuestro. Esto se debe a que cuanto mayor sea el valor de σ, mayor varianza encontraremos en nuestras soluciones (ya que estaremos haciendo una combinación de soluciones mayor).

- ***Motivo 4*** : En el cálculo de σ entran en juego las variables C_{max} y C_{min}, provenientes del conjunto de élite. Estos valores tendrán una función de "límites" al acotar el espacio de búsqueda de soluciones, de manera que no podamos perder la calidad de estas. Como a medida que vamos obteniendo poblaciones de individuos con mayor calidad, la diferencia entre C_{max} y C_{min} será menor, por lo que el valor de σ no cambiará tanto y el espacio de búsqueda quedará acotado a explorar cada vez entre mejores soluciones.

Cabe destacar que σ será un valor muy pequeño; por lo cual he tenido que escalarlo entre un rango de [1, num_individuos] posiciones a modificar; quedándonos por supuesto con la parte entera de este valor.

Una vez reemplazada la población, elegimos aquel de menor función objetivo, siendo este el ***centro de la masa (C_c)***; a modo de implementación, como la población se mantiene ordenada en base a su coste, este será el primer elemento (Paso 3). Se le debería aplicar una búsqueda local (Paso 4) pero como se ha comentado anteriormente, por ahora obviaremos este paso y lo utilizaremos en la mejora del algoritmo.

El conjunto de élite se actualizará en el (Paso 5), donde la mejor solución (el centro de la masa) será añadido sustituyendo la peor solución del conjunto. De esta manera, el algoritmo mantendrá el elitismo y este conjunto albergará cada vez soluciones mejores.

En cada iteración, la búsqueda convergerá gradualmente a una única solución reduciendo sucesivamente el tamaño de la población en el (Paso 6); donde eliminaremos en cada iteración las 10 peores soluciones hasta que solo quede una única solución.

Un nuevo Big Bang comenzará (Paso 7) volviendo al primer paso, donde se generará la nueva población con el método ya descrito. Este proceso se repetirá hasta que se llegue al criterio de parada, que, para mantener la correlación con el resto de prácticas, será las 100000 evaluaciones de la función objetivo. Finalmente, devolveremos la mejor solución encontrada (Paso 8).

Se muestra una tabla con los valores predefinidos utilizados y un pseudocódigo del algoritmo adaptado a nuestro problema:

Parámetro	***Valor***
long_poblacion	***Tamaño de la población = 50 (soluciones factibles)***
NUM_MAX_EVAL	***Número de evaluaciones de la f.objetivo = 100000***
α	***Tasa de reducción = 0.8***
Conjunto de élite	***Tamaño del conjunto de élite = 10***
num_vecinos	***Número de vecinos generados para cada solución = 5***
Elitismo	***La última población está forzada a ser siempre la mejor***

```
def BB_BC(datos, restricciones, n ,d, long_poblacion).:
        num_eval ← 0 , num_vecinos ← 5, soluciones_malas ← 10, tam_elite_pool ←10
        elite_pool ← []

        poblacion ←  generar_poblacion(long_poblacion, k,n,d,elite_pool)
        num_eval ← evaluar_poblacion(poblacion, long_poblacion, num_eval.....)
        sort(poblacion)

        centre_mass ← poblacion[0].valores_asignados

        # Rellenamos la elite pool. Como la población está ordenada de mejor a peor,
        # se eligen los tam_elite_pool primeros valores de la población
        for i in (tam_elite_pool):
                elite_pool[i] ← poblacion[i]
        c_max ← elite_pool[-1].vf_objetivo # El peor
        c_min ← elite_pool[0].vf_objetivo # El mejor

        while(num_evaluaciones  < NUM_MAX_EVAL):
                while len(poblacion) > 1:

                        # Generación de vecinos
                        for i in(long_poblacion):
                                mejor_valor_actual ← inf

                                for i in range(num_vecinos):
                                        num_elem ← 0
                                        hijo ← StellarObject(n,d)
                                        posicion← random(1,n-1)
                                        sigma ← calcula_sigma(...) #Usando la fórmula
                                        hijo.valores_asignados ← centre_mass

                                        while num_elem < sigma:
                                                hijo.valores_asignados[posicion] ←
                                                        poblacion[i].valores-
asignados[posicion]
                                                posicion ← (posicion + 1)%n
                                        evaluar(hijo)
                                        num_eval++

                                        if hijo.vf_objetivo < mejor_valor_actual:
                                                mejor_valor_actual ← hijo.vf_objetivo
                                                mejor_hijo ← hijo
                                        # Sustituimos a cada padre por su mejor
                                        # descendiente
                                        nueva_poblacion.añade(mejor_hijo)
                                        reparar(nueva_poblacion)
                                        sort(nueva_poblacion)
```

```
                    # Actualizamos el centro de la masa
                    centre_mass ← nuvea_poblacion[0].valores_asig
                    # Actualizamos el conjunto de elite (ELITISMO)
                    elite_pool[-1] ← centre_mass

                    si len(nueva_poblacion) > 1:
                        nueva_poblacion ←
                            nueva_poblacion[:len(nueva_poblacion) -
                                (soluciones_malas)]

                si num_eval < NUM_MAX_EVAL:
                    # Volvemos al paso 1 y generamos una nueva poblacion a
                    # partir del conjunto de elite
                    poblacion ←  generar_poblacion(...)
                    num_eval ← evaluar_poblacion(...)
                    sort(poblacion)

    return elite_pool[0] # La mejor solución
```

5. Big Bang Big Crunch + Búsqueda Local

Como ya se ha comentado con anterioridad, realmente la Búsqueda Local no corresponde a ninguna mejora en el BB-BC original puesto que este ya es una hibridación. Sin embargo, la omitimos de primeras para poder realizar el análisis adecuándome a los apartados que exigía el proyecto.

En base a que en la literatura no se habla de ninguna búsqueda local, he decidido utilizar la Búsqueda Local de las prácticas 1 y 3; motivo que justificaré más adelante en el análisis de resultados (Apartado 8).

```
def BusquedaLocal(datos, restricciones, n, d, solucion, lambda):
    num_evaluaciones ← 0
    hay_cambio ← True

    mientras que (hay_cambio) && (num_iteraciones < MAX_ITERACIONES):
        hay_cambio ← False
        # EXPLORACIÓN DEL VECINDARIO Y OPERADOR GENERADOR DE
        # VECINO
        # Realizaremos la técnica del cambio de cluster para generar todos los
        # posibles vecinos, cambiando de cluster cada elemento de la
        # solución
        for i ∈ {0...n}:
            for j i ∈ {0...k}:
                Si j ! = v_solucion[i]: #Si no es el cluster en el que ya está
```

```
                    vecinos  ← v_solucion

                    # Generamos el vecindario asignando a cada
                    # elemento un cluster nuevo
                    vecinos.back()[i] = j

                    #A no ser que el cluster se quede vacío, donde
                    # generaríamos una solución infactible y por tanto,
                    # no tendremos en cuenta este vecino borrándolo
                    si cluster_vacio():
                            borrar vecino

                    #Para no coger siempre los primeros vecinos:
                    shuffle(vecinos)


            #Comparamos el valor de la función objetivo para la solución
            #actual con el de cada vecino hasta encontrar uno que sea
            # mejor.
            #Como nos basamos en una búsqueda local de EL PRIMERO EL
            # MEJOR, en cuanto encontremos una solución que mejore la
        # actual, nos quedaremos con ella
            for v ϵ vecinos:
                    centroide_vecino ← recalcular_centroides(vecino[v]...)
                    sol1  ←  funcion_objetivo(vecino)
                    sol2  ←  funcion_objetivo(v_solucion)

                    si sol1 < sol2
                            v_solucion ←  v
                            hay_cambio ← TRUE
                            break

return v_solucion
```

6. Manual de usuario

Para ejecutar la práctica solo debe indicarse qué conjunto de datos se desea ejecutar.

Tras indicar el conjunto de datos, se ejecutarán ambos conjuntos de restricciones 5 veces.

NO HACE FALTA INDICAR LAS SEMILLAS.

Las semillas utilizadas: (1,3,2,7, 5) han sido guardadas en un vector, de manera que por cada ejecución se seleccionará una semilla diferente.

Esta práctica se ha implementado en python3.

Para ejecutar escribir en terminal: python3 proyecto.py

** Cabe destacar que en el programa ejecutable se ha comentado el código con el que se realizan de manera automática las tablas y gráficas proporcionadas para el análisis de resultados.

7. Tablas de resultados

7.1. Big Bang Big Crunch

Resultados obtenidos en el BB-BC con 10% de restricciones.

	Iris				Ecoli				Rand				Newthyroid			
	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	0,67	0,00	0,66	215,98	37,20	1134,00	67,42	866,01	0,71	0,00	0,71	228,41	13,80	18,00	14,45	375,82
Ejecución 2	0,67	0,00	0,66	214,65	37,18	1110,00	66,76	872,29	0,71	0,00	0,71	227,53	13,83	6,00	14,05	378,48
Ejecución 3	0,67	0,00	0,66	215,07	23,81	1178,00	69,89	872,27	0,71	0,00	0,71	227,11	10,80	118,00	15,11	380,99
Ejecución 4	0,67	0,00	0,66	214,99	38,59	1016,00	65,32	898,04	0,71	0,00	0,71	227,66	13,83	6,00	14,05	377,93
Ejecución 5	0,67	0,00	0,66	215,32	36,53	1003,00	63,61	877,31	0,71	0,00	0,71	227,05	13,83	136,00	14,05	380,29
Media	0,67	0,00	0,66	215,20	34,66	1088,20	66,60	877,18	0,71	0,00	0,71	227,55	13,22	56,80	14,34	378,70

Resultados obtenidos en el BB-BC con 20% de restricciones.

	Iris				Ecoli				Rand				Newthyroid			
	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	0,66	0,00	0,66	291,25	36,87	2354,00	68,24	1104,51	0,71	0,00	0,71	274,76	10,83	231,00	15,05	492,60
Ejecución 2	0,66	0,00	0,66	291,27	35,63	2439,00	68,14	1107,90	0,71	0,00	0,71	273,73	10,85	235,00	10,85	493,51
Ejecución 3	0,66	0,00	0,66	290,71	36,28	2090,00	64,13	1138,96	0,71	0,00	0,71	275,72	14,29	78,00	15,71	489,86
Ejecución 4	0,66	0,00	0,66	290,92	37,71	2112,00	65,86	1121,87	0,71	0,00	0,71	274,27	13,65	218,00	17,63	488,71
Ejecución 5	0,66	0,00	0,66	291,51	35,40	1895,00	60,65	1144,65	0,71	0,00	0,71	275,74	11,60	233,00	15,86	491,16
Media	0,66	0,00	0,66	291,13	36,38	2178,00	65,40	1123,58	0,71	0,00	0,71	274,84	12,24	199,00	15,02	491,17

7.2 Big Bang Big Crunch + Búsqueda Local

Resultados obtenidos en el BB-BC + BL con 10% de restricciones.

	Iris				Ecoli				Rand				Newthyroid			
	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	0,66	0,00	0,66	217,09	22,08	70,00	23,95	969,64	0,71	0,00	0,71	231,73	13,83	6,00	14,05	377,42
Ejecución 2	0,66	0,00	0,66	216,45	21,94	84,00	24,18	950,27	0,71	0,00	0,71	228,46	10,85	98,00	14,43	387,45
Ejecución 3	0,66	0,00	0,66	216,55	22,29	57,00	23,81	994,46	0,71	0,00	0,71	228,06	10,87	96,00	14,37	385,49
Ejecución 4	0,66	0,00	0,66	216,28	22,12	68,00	23,94	868,60	0,71	0,00	0,71	229,10	10,81	113,00	14,94	381,87
Ejecución 5	0,66	0,00	0,66	215,93	21,92	68,00	23,74	906,58	0,71	0,00	0,71	227,66	13,83	6,00	14,05	377,79
Media	0,66	0,00	0,66	216,46	22,07	69,40	23,92	937,91	0,71	0,00	0,71	229,00	12,04	63,80	14,37	382,00

Resultados obtenidos en el BB-BC + BL con 20% de restricciones.

	Iris				Ecoli				Rand				Newthyroid			
	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	0,66	0,00	0,66	291,78	21,84	134,00	23,66	1104,51	0,71	0,00	0,71	278,80	14,28	0,00	14,28	489,81
Ejecución 2	0,66	0,00	0,66	290,33	21,90	182,00	24,33	1158,64	0,71	0,00	0,71	277,80	14,28	0,00	14,28	498,04
Ejecución 3	0,66	0,00	0,66	293,29	21,83	148,00	23,91	1202,55	0,71	0,00	0,71	278,74	14,28	0,00	14,28	494,24
Ejecución 4	0,66	0,00	0,66	291,26	21,83	191,00	24,38	1204,30	0,71	0,00	0,71	274,27	14,28	0,00	14,28	496,89
Ejecución 5	0,66	0,00	0,66	293,90	22,10	154,00	24,15	1182,46	0,71	0,00	0,71	273,93	14,28	0,00	14,28	499,56
Media	0,66	0,00	0,66	292,11	21,90	161,80	24,09	1170,49	0,71	0,00	0,71	276,71	14,28	0,00	14,28	495,71

7.3. Tabla global comparativa

Resultados globales con 10% de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	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	0.14	55,40	7,55	4,39	8,13	448,40	645,61	1652,94	0.16	45,80	8,54	3,64	x	x	x	x
BL	0,11	0,00	0,11	36,13	7,69	709,20	1015,93	2388,08	0,13	0,00	0,13	34,72	x	x	x	x
COPKM-Arreg	1,57	215,00	2,86	0,26	37,24	537,40	51,56	13,93	1,47	164,40	2,64	0,28	19,7	161,00	25,58	0,752
BL-Arreglado	0,67	0,00	0,67	4,14	22,39	55,40	23,86	156,64	0,71	0,00	0,71	3,48	11,6	118,00	15,91	11,73
AGG-UN	0,66	0,00	0,66	211,75	25,23	528,00	39,30	868,21	0,71	0,00	0,71	235,07	13,83	6,00	14,05	398,07
AGG-SF	0,66	0,00	0,66	212,76	27,61	418,00	38,75	875,07	0,71	0,00	0,71	236,11	13,83	6,00	14,05	403,91
AGE-UN	0,66	0,00	0,66	211,71	22,24	45,00	23,44	926,75	0,71	0,00	0,71	232,64	13,83	6,00	14,05	421,17
AGE-SF	0,66	0,00	0,66	210,99	21,41	103,00	24,14	985,57	0,71	0,00	0,71	231,97	13,83	6,00	14,05	430,12
AMI0-1.0	0,66	0,00	0,66	209,66	41,48	1377,00	78,18	898,825	1,34	240,00	3,07	220,87	15,33	61,00	17,55	376,12
AMI0-0.1	0,66	0,00	0,66	214,27	33,78	675,00	51,77	890,42	0,71	0,00	0,71	231,55	13,83	6,00	14,05	390,36
AMI0-0.1MEJ	0,66	0,00	0,66	215,77	29,43	682,00	47,61	891,21	0,71	0,00	0,71	234,81	13,83	6,00	14,05	390,72
ES	0,66	0,00	0,66	24,63	21,48	47,40	22,74	452,35	0,71	0,00	0,71	17,61	13,83	6,00	14,05	143,28
BMB	0,67	0,00	0,66	44,67	21,97	136,80	25,09	958,46	0,71	0,00	0,71	39,8	13,83	6,00	14,05	118,87
ILS	0,66	0,00	0,66	24,79	35,28	77,80	23,67	947,45	0,71	0,00	0,71	23,21	13,83	6,00	14,05	56,48
ILS-ES	0,68	0,00	0,66	84,69	36,82	163,80	26,06	998,13	0,79	0,00	0,71	83,25	13,83	6,00	14,05	224,75
BB-BC	0,67	0,00	0,66	215,2	34,66	1088,20	66,60	877,18	0,71	0,00	0,71	227,55	13,22	56,80	14,34	378,7
BB-BC + BL	0,66	0,00	0,66	216,46	22,07	69,40	23,92	937,91	0,71	0,00	0,71	229,01	12,04	63,80	14,37	382,01

Resultados globales con 20% de restricciones

	Iris				Ecoli				Rand				Newthyroid			
	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	0.12	113,40	7,95	3,29	9,97	386,40	288,68	1652,89	0.17	106,00	10,17	3,39	x	x	x	x
BL	0,11	0,00	0,11	35,88	7,73	1383,60	1005,71	2250,57	0,13	0,00	0,13	29,42	x	x	x	x
COPKM-Arreg	0,76	62,40	0,95	0,53	43,13	644,40	51,71	14,07	1,38	217,40	2,15	0,49	18,61	250,40	23,17	1,45
BL-Arreglado	0,67	0,00	0,67	4,91	21,83	162,20	23,99	290,89	0,72	0,00	0,72	4,37	14,28	0,00	14,28	15,6
AGG-UN	0,66	0,00	0,66	210,65	25,66	617,00	33,89	1173,53	0,71	0,00	0,71	236,97	14,28	6,00	14,28	506,64
AGG-SF	0,66	0,00	0,66	212,55	21,15	336,00	25,62	1191,19	0,71	0,00	0,71	235,79	14,28	6,00	14,28	510,26
AGE-UN	0,66	0,00	0,66	213,48	22,1	162,00	24,26	1245,65	0,71	0,00	0,71	236,12	14,28	6,00	14,28	535,74
AGE-SF	0,66	0,00	0,66	214,22	21,79	181,00	24,20	1241,58	0,71	0,00	0,71	234,82	14,28	6,00	14,28	542,26
AM10-1.0	0,66	0,00	0,66	207,64	40,33	2706,00	76,39	1163,33	1,34	242,00	3,07	220,79	13,52	61,00	14,80	484,13
AM10-0.1	0,66	0,00	0,66	215,33	29,51	746,00	39,45	1169,61	0,71	0,00	0,71	232,92	10,81	6,00	14,96	497,74
AM10-0.1MEJ	0,66	0,00	0,66	215,98	29,92	883,00	41,68	1170,24	0,71	0,00	0,71	232,61	14,28	6,00	14,28	498,53
ES	0,66	0,00	0,66	35,42	21,48	47,40	22,74	448,59	0,71	0,00	0,71	17,77	14,28	6,00	14,28	189,28
BMB	0,67	0,00	0,66	50,33	22,09	145,20	25,96	949,15	0,71	0,00	0,71	43,96	14,28	6,00	14,28	136,42
ILS	0,66	0,00	0,66	29,86	31,02	78,40	23,89	965,47	0,71	0,00	0,71	26,4	14,28	6,00	14,28	93,14
ILS-ES	0,68	0,00	0,66	98,2	36,82	163,80	26,06	998,82	0,71	0,00	0,71	88,57	14,28	6,00	14,28	336,79
BB-BC	0,66	0,00	0,66	291,13	36,18	2178,00	65,40	1123,58	0,71	0,00	0,71	227,55	12,24	199,01	15,02	491,17
BB-BC+BL	0,66	0,00	0,66	291,13	21,9	161,80	24,09	1170,49	0,71	0,00	0,71	276,71	14,28	6,00	14,28	495,71

8. Análisis global de resultados y mejoras

Para analizar el rendimiento sobre los algoritmos, se han ejecutado sobre cuatro conjuntos de datos, uno más que en la práctica anterior:

- ***Iris***: Contiene información sobre características de tres tipos de flor de Iris. Tiene tres clases (k = 3).

- ***Ecoli***: Contiene medidas sobre ciertas características de diferentes tipos de células. Tiene ocho clases (k=8).

- ***Rand***: Conjunto de datos artificial. Contiene tres clases (k = 3).

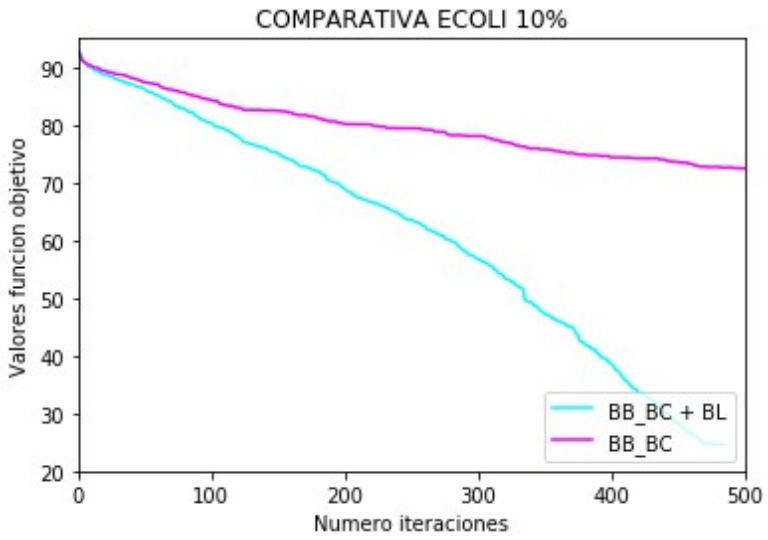
- ***Newthyroid***: Contiene medidas cuantitativas tomadas sobre la glándula tiroides de 215 pacientes. Tiene tres clases (k = 3).

Compararemos ambos algoritmos basándonos en sus capacidades para obtener soluciones de calidad, el número de restricciones no satisfechas, y rapidez de estos, comenzando por comparar los algoritmos genéticos.

Como podemos observar, en los conjuntos *Iris* y *Rand* se llega al óptimo en todos los algoritmos, por lo que se reducirá este análisis a hablar del conjunto de datos *Ecoli*, el cual es el más complejo, y a comentar brevemente *Newtyroid*.

Comenzamos comentando los resultados obtenidos entre el ***BB-BC*** y el ***BB-BC que utiliza Búsqueda Local.*** Observando las tablas generadas, es evidente la diferencia entre los resultados de ambos algoritmos, obteniendo resultados de poca calidad en el *BB-BC* que mejoran considerablemente al aplicarle una Búsqueda Local. A mi juicio, como en el *BB-BC original*, que es una hibridación, se exige el uso de Búsqueda Local, no es de extrañar que los resultados empeoren si no se emplean.

Para verlo de una forma más intuitiva, se ha generado una gráfica con el conjunto de datos *Ecoli a un 10% de restricciones:*



En esta gráfica se ha representado la curvatura que toma el valor de la función objetivo con respecto al número de iteraciones. En *BB-BC* vemos que el "estancamiento" de sus soluciones hacen que tome un espectro de valores muy reducido; por el contrario, el hecho de aplicar una *Búsqueda Local* hace que las soluciones salgan de los mínimos locales en los que se estancan. Además, como se puede observar, a medida que incrementan las iteraciones, el hecho de explotar mediante esta técnica soluciones de calidad hace que cada vez se obtengan soluciones mejores y que, por ende, los valores tomen una curvatura más notoria.

Esto nos demuestra que, a pesar de una buena exploración del vecindario de los individuos, es fundamental realizar la explotación de soluciones de calidad para poder encaminar a un algoritmo a una solución mejor.

En base a nuestro algoritmo comparativo, ***Greedy***, es obviamente mejor; aunque no sea algo complicado de lograr puesto que este ni siquiera consigue llegar al óptimo en el conjunto de datos más sencillo, *Iris*.

Con respecto a los *algoritmos basados en trayectorias*, se observa que nuestro algoritmo (con *Búsqueda Local*) le hace competencia sin problema alguno. Se obtienen resultados mejores que la ***Búsqueda Multiarranque (BMB)*** y que la ***Búsqueda Local Iterativa con Enfriamiento Simulado (ILS-ES)***, lo cual es un gran punto a favor.

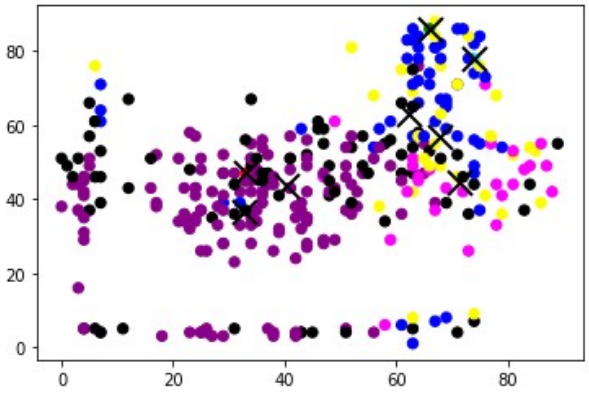
A pesar de que no llega a superar al ***Enfriamiento Simulado (ES)*** y ***Búsqueda Local Iterativa (ILS)***, los cuales son los dos mejores algoritmos hasta ahora, consigue alcanzar un valor medio de su función objetivo muy similar, variando únicamente unas décimas. La mayor desventaja que se encuentra con respecto a los *algoritmos de búsqueda por trayectorias* es la gran diferencia de tiempo de ejecución, lo cual es totalmente comprensible ya que estos algoritmos trabajan con una solución la cual se va modificando

sucesivamente y *Big Bang Big Crunch* con un conjunto (población) de soluciones.

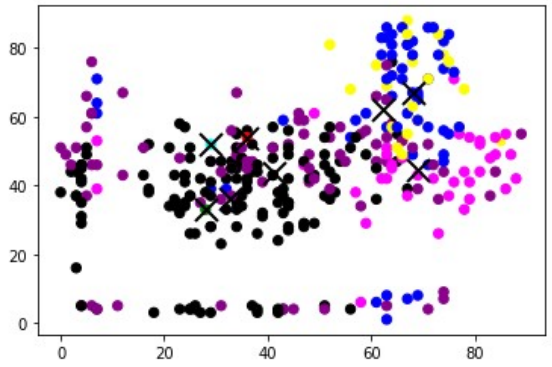
Se ha preferido comparar en último lugar los ***algoritmos genéticos*** *y* ***meméticos,*** debido a su estrecha similitud con ellos. Todos son algoritmos basados en poblaciones, lo que indica que el tiempo de ejecución sea similar entre ellos pero más elevado con respecto a otros algoritmos.

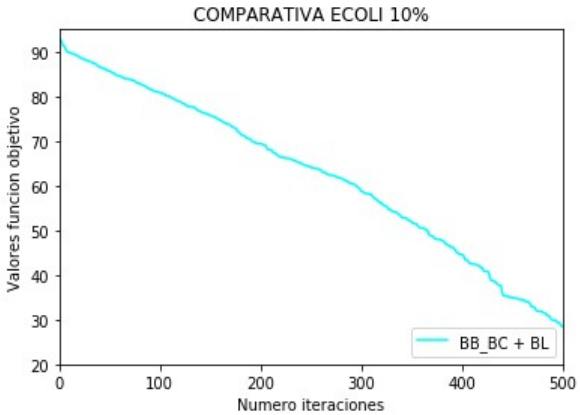
A pesar de que *BB-BC* se asemeje más a un *algoritmo memético* que a un genético ya que ambos casos son una hibridación, este obtiene resultados notablemente mejores que los *meméticos*, los cuales es evidente que no se adaptaron bien al nuestro problema en general. Además, es muy positivo el hecho de que se haya obtenido resultados algo mejores que la mayoría de los genéticos, quedando solamente por delante ***AGE-UN***; sin embargo, cabe destacar que para conseguir esto hemos tenido que hacer uso de la hibridación, ya que sin ella se obtienen resultados bastante peores.

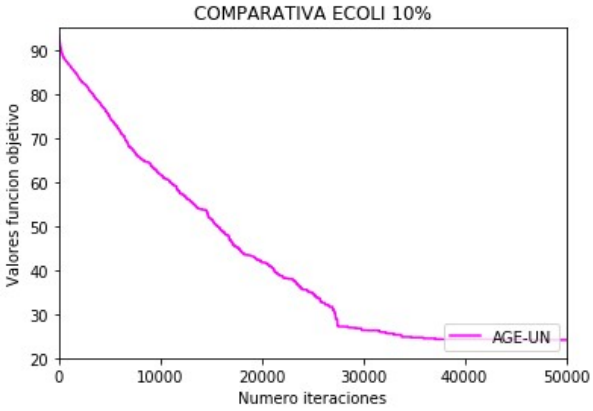
BB-BC + BL , Ecoli 10% restricciones



AGE-UN , Ecoli 10% restricciones







En ambas asignaciones de clústeres encontramos agrupaciones similares; donde algunos clústeres contienen a la gran mayoría de datos, y otros, como en el caso de los clústeres *(cyan-1, red-4 y green-6)*, que tienen muy pocos elementos (o incluso un único punto), haciendo difícil ver la asignación. Al tratar con el conjunto *Ecoli*, se pueden observar visualmente que cómo algunos puntos que deberían estar más agrupado, no lo están (pudiendo ser esto razón del incremento del valor *"infeasibility"*).

En base a las gráficas, a pesar de que ambos algoritmos lleguen a un valor similar, el hecho de que *AGE-UN* sea un poco mejor, no se debería tener realmente en consideración; esto se debe a que, en comparación, *BB-BC* con Búsqueda Local alcanza un valor bueno en apenas 500 iteraciones, en contraste a las aproximadamente 50000 (la mitad del número de evaluaciones de la función objetivo) del *AGE-UN*.

De nuevo, observamos la importancia de una buena explotación de soluciones, ya que para el algoritmo genético, necesita realizar una gran exploración del entorno hasta poder disminuir sus resultados.

Tras compararlo con todos los algoritmos, no cabe duda de que *BB-BC con Búsqueda Local* ha obtenido buenos resultados. *Big Bang Big Crunch* es una metaheurística que ha combinado la larga búsqueda por el espacio de búsqueda y una explotación agresiva de las mejores soluciones; por lo que tenemos la seguridad de generar soluciones de calidad (o las óptimas).

Aún así, el usar *Búsqueda Local* no es el único factor determinante en la efectividad de la metaheurística. Una buena exploración de soluciones factibles por el espacio de búsqueda es fundamental y esto se ha conseguido gracias al Conjunto de élite y al reducir sucesivamente el tamaño de la población ya que, de esta manera las siguientes poblaciones, al partir del *Conjunto de Élite*, se generarán con una distribución generalmente ordenada, puesto que nos aseguramos que sus costes no difieran mucho entre los distintos individuos de la población. Esto también permite que el algoritmo escape de mínimos locales y que se reinicie la búsqueda de una manera efectiva, ya que partir de una solución inicial con un coste alto es más propenso a converger rápidamente que partiendo de una solución inicial de menor calidad.

Como resultado de todo este análisis, podemos concluir que *Big Bang Big Crunch* se ha adaptado adecuadamente a nuestro problema, llegando a los óptimos de *Iris* y *Rand*; obteniendo buenos resultados (aunque no siempre óptimos) en *Newthyroid*, y siendo uno de los mejores algoritmos con respecto a *Ecoli*.

- Experimentación:

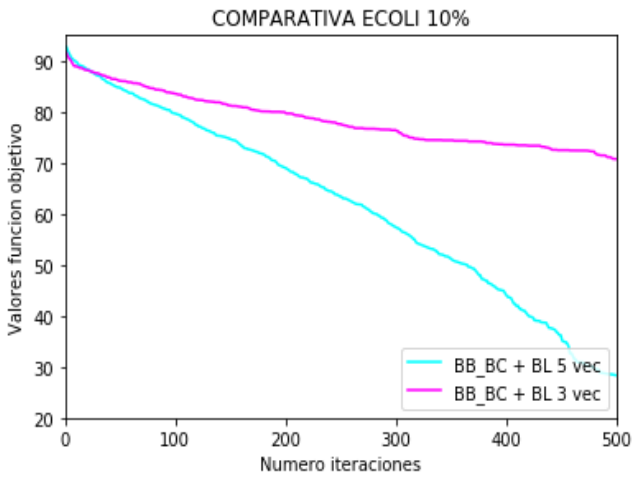
La experimentación se ha basado en un ajuste de los hiperparámetros del problema en busca de mejores resultados. Por ello, todo lo explicado anteriormente es la mejor combinación obtenida que se ha conseguido.

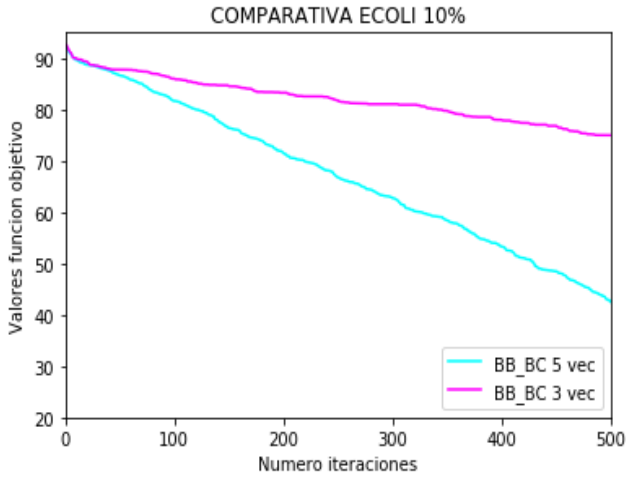
Compararemos estas mejoras para el conjunto Ecoli con un 20% de restricciones.

- ***Número de vecinos***: Inicialmente, en la fase *Big Crunch* del algoritmo se probó a generar 3 vecinos por cada solución. En la implementación final se generan 5 vecinos.

Los resultados medios para comparar esta mejora son los siguientes:

	Desviación	***Infeasibility***	***F.Objetivo***	***Tiempo***
BB-BC 5 vec.	**34,66**	**1088,20**	**66,60**	**877,18**
BB-BC 3 vec.	**37,82**	**1060,40**	**51,94**	**974,97**
BB-BC+BL 5 vec.	**21,90**	**161,80**	**34,09**	**1170,49**
BB-BC+BL 3 vec.	**37,37**	**926,40**	**49,71**	**1006,33**





Cada solución de la población genera vecinos y se reemplaza por el mejor obtenido. Como se puede ver, el ampliar el entorno de exploración para cada solución conlleva una mejoría en los resultados, la cual es más evidente a medida que se aumentan las iteraciones.

Llama la atención que, usando solamente 3 vecinos, los resultados de *BB-BC y BB-BC con Búsqueda Local* no difieren mucho, sin embargo, el valor de la función objetivo con 5 vecinos se reduce casi a la mitad entre ambos algoritmos, lo que nos puede llevar a pensar en que aumentar el espacio de búsqueda hace que se explore soluciones *bastante* mejores.

- ***Algoritmo de Búsqueda Local***: En un principio, al tratarse de un algoritmo basado en poblaciones, decidí usar la ***Búsqueda Local suave*** de la segunda prácticas. Estas son los diferencias encontradas:

	Desviación	***Infeasibility***	***F.Objetivo***	***Tiempo***
B.Local suave	**21,90**	**161,80**	**34,09**	**1170,49**
Búsqueda Local	**36,33**	**118,20**	**66,12**	**752,89**

Claramente la *Búsqueda Local Suave* no causa ninguna mejora (incluso empeora levente en ciertas ejecuciones) al *Big Bang Big Crunch*. La *Búsqueda Local Suave* solo genera vecinos de forma aleatoria y asigna a esos elementos el mejor valor posible; que por lo que se observa, no es una táctica muy eficaz en este algoritmo (ni en los genéticos implementados, en base a los resultados de los *meméticos*). Por otro lado, la Búsqueda Local elegida explora todo el vecindario quedándonos con el primer mejor; esta exploración más agresiva es la que realmente acaba marcando la diferencia de los resultados.

Si se hubiese elegido la táctica de aplicar la *Búsqueda Local Suave* a un mayor número de soluciones buenas, por ejemplo, al *Conjunto de Élite* entero, quizás se hubiesen obtenido mejores resultados; sin embargo, como se le da especial importancia a quedarnos con la mejor solución (el centro de la masa, que es la solución que queda cuando reducimos la población entera), se consideró más adecuado experimentar con otra *Búsqueda Local* diferente.

9. Bibliografía

- H. M. Genc, A. K. Hocaoglu, "Bearing-Only Target Tracking Based on Big Bang – Big Crunch Algorithm," In: The Proceedings of the 3rd International Multi-Conference on Computing in the Global Information Technology (ICCGI '08). DOI 10.1109/ICCGI.2008.53, July 27-Aug. 1, pp. 229 – 233, 2008.

- O. K. Erol, I. Eksin, "A new optimization method: Big Bang–Big Crunch," Advances in Engineering Software, Elsevier, vol. 37, pp. 106–111, 2006.

- M. Kripka, M. L. Kripka, "Big Crunch Optimization Method," EngOpt 2008 - International Conference on Engineering Optimization Rio de Janeiro, Brazil, June, 2008.