Metaheurística

Práctica 3.b:

Búsquedas por Trayectorias para el Problema del Agrupamiento con Restricciones

Curso 2019-2020

Javier Gálvez Obispo
javiergalvez@correo.ugr.es
Grupo 2 Jueves 17:30 – 19:30

Índice

1. Descripción del problema

Nos encontramos ante un problema de agrupamiento o clustering, donde nuestro objetivo es clasificar los datos que nos dan de acuerdo a posibles caractarísticas comunes entre ellos.

Además, en nuestro caso nos encontramos con restricciones de instancia a la hora de agrupar los datos, es decir, dada un pareja de instancias del conjuto de datos puede, o no, haber una restricción entre ellos del tipo *Must-Link*, los dos elementos deben estar en el mismo cluster, o del tipo *Cannot-Link*, si las dos instancias deben encontrarse en clusters diferentes.

Vamos a tratar estas restricciones como restricciones débiles, lo que quiere decir que nuestro objetivo será minimizar el número de restricciones que no se cumplen en vez de encontrar una solución en la que el número de restricciones incuplidas sea cero.

Nuestro objetivo es resolver este problema dados cuatro conjutos de datos diferentes junto a dos listas de restricciones por conjuto de datos, una con un 10% de restricciones y otra con un 20%.

En esta práctica vamos a resolver este problema utilizando técnicas de búsqueda basadas en trayectorias tanto simples, búsqueda local (BL) y enfriamiento simulado (ES), como compuestas, búsqueda multiarranque basica (BMB) y búsqueda local reiterada (ILS). Probaremos dos ILS distintas, una que haga uso de la BL y otra que utilice ES como algoritmo de búsqueda.

2. Elementos en común entre los algoritmos

2.1. Esquema de representación de soluciones

La representación de una posible solución a nuestro problema se hace mediante un vector de longitud *N*, tamaño del conjuto de datos, en el que se almacenan enteros entre 0 y *k*, el número de clusters del problema. Entonces, en la posición *i* de nuestro vector solución *S* estará reflejado el cluster *c* al que pertenece el punto *i* del conjuto de datos. Cada vector solución se guarda junto con el valor resultado de la función objetivo en un ***struct*** *Solución* que facilita el tratamiento de las soluciones.

Struct *Solución* **contains**
    vector<int> S
    float f = -1 // El valor de la función objetivo se inicializa a -1
end

2.2. Función objetivo

La función objetivo se define como:

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

donde $\vec{C}$ es la desviación general de la solución S, *infeasibility* el número de restricciones incumplidas y λ es un parámetro de escalado.

2.2.1. Cálculo de la desviación general

Para calcular la desviación general de una solución primero debemos calcular los centroides μ_i (vector promedio) asociado a cada cluster c_i.

function *calcula_centroides*

Input: Solución *S*, conjunto de datos *X*, número de clusters *k*

double centroides[*k*][*X*[0].size] = [[0, …, 0], …, [0, …, 0]]
int puntos_por_cluster[*k*] = [0, 0, …, 0]

for i **in** {0, 1, …, *X*.size - 1} **do**:
    int cluster = *S*[i]
    puntos_por_cluster[cluster] += 1
    centroides[cluster] = centroides[cluster] + *X*[i]
end for
return centroides / puntos_por_cluster

Calculamos ahora la distancia media intra-cluster, siendo ésta la media de las distancias de las instancias de cada cluster con su centroide asociado.

function *calcula_media_intracluster*

Input: Solución *S*, conjunto de datos *X*, número de clusters *k*

double media_intracluster[*k*] = [0, 0, …, 0], centroides[*k*][*X*[0].size] = *calcula_centroides(S, X, k)*
int puntos_por_cluster[k] = [0, 0, …, 0]

for i **in** {0, …, *X*.size - 1} **do**:
    int cluster = *S*[i]
    puntos_por_cluster[cluster] += 1
    media_intracluster[cluster] += distancia(*X*[i], centroides[cluster])
    // distancia(x, y) calcula la distancia euclidea entre 2 puntos dados
end for
return distancia_media_intracluster / puntos_por_cluster

Y una vez tenemos las distancias medias intra-cluster ya podemos calcular la desviación general de la solución como la media de las distancias recién calculadas.

function *calcula_c*

Input: Solución *S*, conjunto de datos *X*, número de clusters *k*

double c = 0**,** media_intracluster[*k*] = *calcula_media_intracluster(S, X, k)*

```
for distancia in media_intracluster do:
        c = c + distancia
end for
return c / k
```

En el código implementado estas funciones están unidas en una única función

2.2.2. Cálculo de la *infeasibility*

Para calcular la *infeasibility* debemos contar el número de restricciones incumplidas en la solución S, es decir, las veces que se dan los hechos:

- dos instancias en cluster distintos con una restricción *Must-Link*
- dos instancias en un mismo cluster con una restricción *Cannot-Link*

```
function calcula_infeasibility

Input: Solución S, conjuto de restricciones R.

int restricciones_incumplidas = 0

for i in {0, 1, …, S.size - 2} do:
        for j in {i+1, i+2, …, S.size - 1} do:
        if (R_ij == 1) do:
            Si X[i] y X[j] no están en el mismo cluster => restricciones_incumplidas += 1
        elif (R_ij == -1) do:
            Si X[i] y X[j] están en el mismo cluster => restricciones_incumplidas += 1
        end for
end for
return restricciones_incumplidas
```

2.2.3. Cálculo de λ

Definimos λ como el cociente entre la distancia de los puntos más alejados del conjuto y el número de restricciones presentes en el problema.

```
function calcula_lambda

Input: Conjuto de datos X, conjuto de restricciones R.

double max_dist = 0
int num_restricciones = 0
for i in {0, 1, …, X.size - 2} do:
        for j in {i+1, i+2,  …, X.size - 1} do:
            double distancia = distancia(X[i], X[j])
            if (distancia > max_dist) max_dist = distancia
            if (R[i][j] != 0) num_restricciones += 1
        end for
end for
```

return lambda = max_dist / num_restricciones

Entonces, en el código de la función objetivo sólo tenemos que llamar a éstas funciones que acabamos de definir

function *calcula_f*

Input: Solución *S*, Conjuto de datos *X*, conjuto de restricciones *R*, número de clusters *k*.
return *calcula_c(S, X, k) + calcula_infeasibility(S, R) * calcula_lambda(X, R)*

En el código implementado solo se llama una vez a *calcula_lambda(X, R)* por conjunto de datos.

2.3. Generación de soluciones aleatorias

Para generar una solución aleatoria generamos *N*, tamaño del conjuto de datos, número aleatorios entre 0 y *k*, el número de clusters del problema y comprobamos que ninguno de los cluster esté vacio.

function *generar_solución_aleatoria*

Input: Tamaño del conjuto de datos *N*, clusters *k*.

Bool cluster_vacio = true
Solución Solución

while cluster_vacio == true **do**:
    cluster_vacio = false
    int cuenta[k] = [0, …, 0]

    for i **in** {0, 1, …, *N* – 1} **do**:
        int cluster = *random_int*(0, *k*)
        Solución.S[i] = cluster
        cuenta[cluster] += 1
    end for

    if 0 **in** cuenta **then** cluster_vacio = true
end while
Solución.f = *calcula_f*(Solución.S)
return S

3. Pseudocódigo de los algoritmos

3.1. Búsqueda local

El pseudocódigo de la búsqueda local utilizada tanto en la búsqueda multiarranque básica como la búsqueda local reiterada es el siguiente.

function *BL*

Input: Solución *Sol*, Conjuto de datos *X*, conjuto de restricciones *R*, número de clusters *k*, máximo de evaluaciones *max_evals*.

```
int evaluaciones = 0
bool mejora = True

while evaluaciones < max_evals and mejora == True do:
        mejora = False

        vector <Int, Int> parejas
        for i in {0, 1, …, X.size} do:
            for j in {0, 1, …, k} do:
                if j != Sol.S[i] then
                    parejas.push(<i, j>)
                end if
            end for
        end for
        parejas = shuffle(parejas)

        int indice = 0
        while indice < parejas.size and mejora == False do:
            int antiguo_cluster = Sol.S[parejas[indice].i]
            Sol.S[parejas[indice].i] = parejas[indice].j
            if antiguo_cluster in Sol.S then
                double nueva_f = calcula_f(Sol.S, X, R, k)
                if nueva_f < Sol.f then
                    Sol.f = nueva_f
                    mejora = True
                else
                    Sol.S[parejas[indice].i] = antiguo_cluster
                end if
            end if
            indice++
        end for
end while
return Sol
```

3.2. Búsqueda Multiarranque Básica

En la BMB lo único que tenemos que hacer es ejecutar *iters* veces la BL con soluciones iniciales distintas y quedarnos con la que nos de mejor resultado.

function *BMB*

Input: Conjuto de datos *X*, conjuto de restricciones *R*, número de clusters *k*, número de iteraciones *iters*, máximo de evaluaciones para la búsqueda local *bl_max_evals*.

Solución mejor_solución, solución_actual
mejor_solución.f = ∞

for i **in** {0, 1, …, *iters*} **do:**
    solución_actual = *generar_solución_aleatoria*(*X*.size, *k*)
    solución_actual = *BL*(solución_actual, *X*, *R*, *k*)
    if solución_actual.f < mejor_solución.f **then**
        mejor_solución = solución_actual
    end if
end for

return mejor_solución

3.3. Enfriamiento Simulado

El esquema de enfriamiento que vamos a implementar es el modelo de Cauchy.

function *ES*

Input: Conjuto de datos *X*, conjuto de restricciones *R*, número de clusters *k*, máximo de evaluaciones *max_evals*, valor de μ, valor de ϕ.

Solución mejor_solución, solución_actual
solución_actual = *generar_solución_aleatoria*(*X*.size, *k*)
mejor_solución.f = solución_actual

double t0 = (μ * solución_actual) / (- log(ϕ)), tf = 0.001, t = t0
int max_vecinos = 10 * *X*.size, max_exitos = 0.1 * max_vecinos, M = *max_evals* / max_vecinos
double beta = (t0 – tf) / (M * t0 * tf)

int vecinos, exitos = 1, enfriamientos = 0
while enfriamientos < M **and** exitos > 0 **do:**
    vecinos = 0, exitos = 0
    while vecinos < max_vecinos **and** exitos < max_exitos **do:**
        int i = *rand_int*(0, *X*.size), cluster_antiguo = solución_actual.S[i], c = cluster_antiguo
        while c == cluster_antiguo **do:**
            c = *rand_int*(0, *k*)
        end while
        solución_actual.S[i] = c
        if antiguo_cluster **in** solución_actual.S **then**
            double nueva_f = *calcula_f*(*Sol*.S, *X*, *R*, *k*),
            double delta_f = nueva_f - solución_actual.f
            if nueva_f < solución_actual.f **or** U(0, 1) < exp(- delta_f / t) **then**
                Sol.f = nueva_f
                exitos++
                if solution.f < best_solution.f **then**

```
                    mejor_solución = solución_actual;
                end if
            else
                solución_actual.S[i] = antiguo_cluster
            end if
        else
            solución_actual.S[i] = antiguo_cluster
        end if
        vecinos++
    end while
    t = t / (1 + beta * t), enfriamientos++
end while

return mejor_solución
```

Si queremos implementar el esquema proporcional cambiamos la forma en la que actualizamos *t* por $t = \alpha\, t$ y la condición del bucle exterior *enfriamientos* < *M* se sustituye por *evals* < *max_evals*, *evals* se actualiza cada vez que llamamos a la función objetivo.

3.4. Búsqueda Local Reiterada

3.4.1. Mutación

En la ILS introducimos diversidad mediante el operador de mutación que modifica un segmento de la solución de forma aleatoria. La longitud de éste segmento depende del tamaño del problema, sólo modificaremos un 10% de la solución.

```
function Mutación-ILS
```

Input: Solución *Sol*, tamaño del segmento *v*, Conjuto de datos *X*, conjuto de restricciones *R*, número de clusters *k*.

```
bool cluster_vacio = True
Solución mutación
while cluster_vacio == True do:
    cluster_vacio = False
    mutación.S.clear()
    int n = X.size, r = rand_int(0, n), c
    int cuenta[k] = [0, …, 0]

    if (r + v) > n then
        int aux = (r + v) % n
        for i in {0, 1, …, n} do:
            if i >= r or i < aux then c = rand_int(0, k)
            else c = Sol.S[i]
            mutación.S.push(c), cuenta[c]++
        end for
```

```
        else
            for i in {0, 1, …, n} do:
                if i >= r and i < (r + v) then c = rand_int(0, k)
                else c = Sol.S[i]
                mutación.S.push(c), cuenta[c]++
            end for
        end if

        if 0 in cuenta then
            cluster_vacio = True
        end if
end while
mutación.f  = calcula_f(mutación.S, X, R,  k)
return mutación
```

3.4.2. Búsqueda Local como algoritmo de búsqueda

```
function ILS-BL
```

Input: número de iteraciones *iters*, número de evaluaciones de la BL *bl_max_evals*, Conjuto de datos *X*, conjuto de restricciones *R*, número de clusters *k*.

```
int v = 0.1 * X.size
Solución mejor_solución, solución
solución_actual = generar_solución_aleatoria(X.size, k)
mejor_solución = solución_actual

for i in {0, 1, …, iters} do:
        solución_actual = BL(solución_actual, X, R, k)
        if solución_actual.f < mejor_solución.f then
            mejor_solución = solución_actual
        end if
        solución_actual = Mutación-ILS(solución_actual, v, X, R, k)
end for
return mejor_solución
```

3.4.3. Enfriamiento Simulado como algoritmo de búsqueda

```
function ILS-ES
```

Input: número de iteraciones *iters*, número de evaluaciones de la BL *bl_max_evals*, Conjuto de datos *X*, conjuto de restricciones *R*, número de clusters *k*.

```
int v = 0.1 * X.size
Solución mejor_solución, solución
solución_actual = generar_solución_aleatoria(X.size, k)
```

mejor_solución = solución_actual

for i **in** {0, 1, …, *iters*} **do**:
    solución_actual = *ES*(solución_actual, *X*, *R*, *k*, 0.3, 0.3)
    if solución_actual.f < mejor_solución.f **then**
        mejor_solución = solución_actual
    end if
    solución_actual = *Mutación-ILS*(solución_actual, v, *X*, *R*, *k*)
end for
return mejor_solución

4. Procedimiento considerado para desarrollar la práctica

Para implementar estos algoritmos se ha utilizado C++ empezando desde cero sin hacer uso de ningún framework. Para realizar una prueba de los algoritmos es necesario compilar el código fuente mediante el uso del makefile. Para ejecutar el programa hay que seguir la siguiente sintaxis:

./test dataset porcentaje-de-restricciones numero-de-clusters semilla algoritmo

un ejemplo de ejecución sería "./test ecoli 10 8 11264 es". Los valores que puede tomar *algoritmo* son: es, bmb, ils-ls e ils-es.

5. Experimentos y análisis de resultados

Se han realizado 5 pruebas por algoritmo utilizando siempre las mismas 5 semillas: 11264, 16438, 75645, 79856 y 96867.

5.1. Resultados Enfriamiento Simulado

Se ha ejecutado el algoritmo ES utilizando el esquema de Cauchy, con μ=ϕ=0.3 y un máximo de 100000 evaluaciones.

	Iris			Rand			Newthyroid			Ecoli		
Seed	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea
11264	0.669	0.669	0	0.716	0.716	0	14.054	13.835	6	24.109	21.318	104
16438	0.669	0.669	0	0.716	0.716	0	14.054	13.835	6	24.005	21.697	86
75645	0.669	0.669	0	0.716	0.716	0	15.037	10.795	116	23.919	22.282	61
79856	0.669	0.669	0	0.716	0.716	0	14.054	13.835	6	23.405	19.220	156
96867	0.669	0.669	0	0.716	0.716	0	14.054	13.835	6	24.402	22.283	79
Media	**0.669**	**0.669**	**0**	**0.716**	**0.716**	**0**	**14.251**	**13.227**	**28**	**23.968**	**21.360**	**97.2**

ES 20% restricciones												
	Iris			Rand			Newthyroid			Ecoli		
Seed	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea
11264	0.669	0.669	0	0.716	0.716	0	15.185	10.889	235	24.004	21.925	155
16438	0.669	0.669	0	0.716	0.716	0	14.287	14.287	0	24.041	21.921	158
75645	0.669	0.669	0	0.716	0.716	0	15.192	10.895	235	23.877	21.838	152
79856	0.669	0.669	0	0.716	0.716	0	15.192	10.895	235	24.041	21.882	161
96867	0.669	0.669	0	0.716	0.716	0	14.287	14.287	0	24.696	21.691	224
Media	**0.669**	**0.669**	**0**	**0.716**	**0.716**	**0**	**14.829**	**12.251**	**141**	**24.132**	**21.851**	**170.0**

Tiempos ES (segundos)								
	Iris		Rand		Newthyroid		Ecoli	
Seed	10%	20%	10%	20%	10%	20%	10%	20%
11264	0.14	0.23	0.13	0.17	0.28	0.44	3.99	4.62
16438	0.17	0.22	0.11	0.10	0.29	1.84	2.62	4.35
75645	0.17	0.15	0.12	0.17	0.30	0.33	2.37	8.61
79856	0.15	0.18	0.12	0.17	0.21	0.37	2.90	6.55
96867	0.14	0.20	0.15	0.14	0.30	2.09	2.02	3.50
Media	**0.15**	**0.20**	**0.13**	**0.15**	**0.28**	**1.01**	**2.78**	**5.53**

5.2. Resultados Búsqueda Multiarranque Básica

En cada ejecución de la búsqueda multiarranque básica se realizan 10 búsqueda locales con soluciones aleatorias. Cada BL puede realizar como mucho 10000 evaluaciones de la función objetivo.

	Iris			Rand			Newthyroid			Ecoli		
Seed	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea
11264	0.669	0.669	0	0.716	0.716	0	14.054	13.835	6	29.085	23.826	196
16438	0.669	0.669	0	0.716	0.716	0	14.054	13.835	6	25.907	22.821	115
75645	0.669	0.669	0	0.716	0.716	0	14.054	13.835	6	24.603	22.322	85
79856	0.669	0.669	0	0.716	0.716	0	14.054	13.835	6	26.549	21.827	176
96867	0.669	0.669	0	0.716	0.716	0	14.054	13.835	6	25.722	22.449	122
Media	**0.669**	**0.669**	**0**	**0.716**	**0.716**	**0**	**14.054**	**13.835**	**6**	**26.373**	**22.649**	**138.8**

BMB 20% restricciones												
	Iris			Rand			Newthyroid			Ecoli		
Seed	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea
11264	0.669	0.669	0	0.716	0.716	0	14.287	14.287	0	25.603	22.370	241
16438	0.669	0.669	0	0.716	0.716	0	14.287	14.287	0	25.408	22.457	220
75645	0.669	0.669	0	0.716	0.716	0	14.287	14.287	0	25.791	22.920	214
79856	0.669	0.669	0	0.716	0.716	0	14.287	14.287	0	24.905	22.598	172
96867	0.669	0.669	0	0.716	0.716	0	14.287	14.287	0	25.245	21.891	250
Media	**0.669**	**0.669**	**0**	**0.716**	**0.716**	**0**	**14.287**	**14.287**	**0**	**25.390**	**22.447**	**219.4**

Tiempos BMB (segundos)								
	Iris		Rand		Newthyroid		Ecoli	
Seed	10%	20%	10%	20%	10%	20%	10%	20%
11264	0.29	0.38	0.32	0.43	1.26	1.87	10.71	15.65
16438	0.27	0.38	0.29	0.36	1.12	1.85	11.18	16.49
75645	0.28	0.41	0.31	0.40	1.12	1.53	11.17	15.54
79856	0.31	0.43	0.30	0.39	1.18	1.72	10.39	15.45
96867	0.29	0.42	0.31	0.37	1.13	1.67	10.50	15.64
Media	**0.29**	**0.40**	**0.30**	**0.39**	**1.16**	**1.73**	**10.79**	**15.75**

5.3. Resultados Búsqueda Local Reiterada

5.3.1. Búsqueda Local como algoritmo de búsqueda

En cada ejecución de la ILS-BL se realizan 10 búsqueda locales, la primera será sobre una solución aleatoria y el resto sobre soluciones mutadas. Cada BL puede realizar como mucho 10000 evaluaciones de la función objetivo.

ILS-BL 10% restricciones												
	Iris			Rand			Newthyroid			Ecoli		
Seed	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea
11264	0.669	0.669	0	0.716	0.716	0	14.054	13.835	6	23.387	21.079	86
16438	0.669	0.669	0	0.716	0.716	0	14.368	10.894	95	23.827	21.198	98
75645	0.669	0.669	0	0.716	0.716	0	14.054	13.835	6	23.547	21.615	72
79856	0.669	0.669	0	0.716	0.716	0	14.384	10.874	96	23.459	21.715	65
96867	0.669	0.669	0	0.716	0.716	0	14.427	10.880	97	23.610	22.241	51
Media	**0.669**	**0.669**	**0**	**0.716**	**0.716**	**0**	**14.258**	**12.063**	**60**	**23.566**	**21.570**	**74.4**

ILS-BL 20% restricciones												
	Iris			Rand			Newthyroid			Ecoli		
Seed	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea
11264	0.669	0.669	0	0.716	0.716	0	14.287	14.287	0	23.741	21.675	154
16438	0.669	0.669	0	0.716	0.716	0	14.287	14.287	0	23.476	21.759	128
75645	0.669	0.669	0	0.716	0.716	0	14.287	14.287	0	23.557	22.001	116
79856	0.669	0.669	0	0.716	0.716	0	15.066	10.880	229	23.627	21.884	130
96867	0.669	0.669	0	0.716	0.716	0	14.287	14.287	0	23.790	21.925	139
Media	**0.669**	**0.669**	**0**	**0.716**	**0.716**	**0**	**14.443**	**13.606**	**45.8**	**23.638**	**21.849**	**133.4**

Tiempos ILS-BL (segundos)								
	Iris		Rand		Newthyroid		Ecoli	
Seed	10%	20%	10%	20%	10%	20%	10%	20%
11264	0.17	0.23	0.18	0.24	0.71	1.17	10.65	15.04
16438	0.17	0.26	0.17	0.25	0.63	1.28	11.10	16.39
75645	0.18	0.26	0.18	0.23	0.61	1.06	10.54	16.35
79856	0.18	0.26	0.17	0.26	0.86	1.07	10.86	15.42
96867	0.16	0.25	0.18	0.27	0.74	1.16	11.00	16.24
Media	**0.17**	**0.25**	**0.18**	**0.25**	**0.71**	**1.15**	**10.83**	**15.89**

5.3.2. Enfriamiento Simulado como algoritmo de búsqueda

En cada ejecución de la ILS-ES se realizan 10 enfriamiento simulados, la primera será sobre una solución aleatoria y el resto sobre soluciones mutadas. Debido a que el esquema de Cauchy no conseguía resultados buenos también se han realizado pruebas con el esquema propocional.

5.3.1.1. Esquema de Cauchy

Utilizamos los mismos valores para μ y ϕ que anteriormente. El único parámetro que modificamos es el máximo de evaluaciones, que ahora será 10000.

ILS-ES Cauchy 10% restricciones												
	Iris			Rand			Newthyroid			Ecoli		
Seed	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea
11264	0.669	0.669	0	0.716	0.716	0	15.062	11.515	97	62.244	36.542	958
16438	0.669	0.669	0	0.716	0.716	0	14.054	13.835	6	63.451	36.031	1022
75645	0.669	0.669	0	0.803	0.731	10	14.054	13.835	6	54.721	33.365	796
79856	0.669	0.669	0	0.716	0.716	0	14.054	13.835	6	63.223	36.743	987
96867	0.669	0.669	0	0.782	0.732	7	14.054	13.835	6	62.708	34.698	1044
Media	**0.669**	**0.669**	**0**	**0.746**	**0.722**	**3.4**	**14.256**	**13.371**	**24.2**	**61.269**	**35.475**	**961.4**

ILS-ES Cauchy 20% restricciones												
	Iris			Rand			Newthyroid			Ecoli		
Seed	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea
11264	0.727	0.689	12	0.716	0.716	0	14.287	14.287	0	57.711	35.282	1672
16438	0.669	0.669	0	0.716	0.716	0	14.287	14.287	0	63.336	36.091	2031
75645	0.723	0.678	14	0.810	0.745	18	15.531	10.814	258	58.632	34.874	1771
79856	0.731	0.674	18	0.782	0.735	13	15.192	10.895	235	61.154	34.687	1973
96867	0.669	0.669	0	0.716	0.716	0	14.287	14.287	0	59.530	34.256	1884
Media	**0.704**	**0.676**	**8.8**	**0.748**	**0.725**	**6.2**	**14.717**	**12.914**	**98.6**	**60.073**	**35.038**	**1866.2**

Como podemos observar los resultados son malos ya que, al ser el valor de max_evals un valor bajo, se realizan muy pocas iteraciones del bucle exterior. Se ha probado ha reducir el valor de max_vecinos y max_exitos pero no se han conseguido buenos resultados porque el algoritmo enfría demasiado rápido.

Tiempos ILS-ES Cauchy (segundos)								
	Iris		Rand		Newthyroid		Ecoli	
Seed	10%	20%	10%	20%	10%	20%	10%	20%
11264	0.10	0.11	0.09	0.13	0.41	0.51	0.43	0.62
16438	0.09	0.12	0.10	0.11	0.32	0.59	0.45	0.60
75645	0.09	0.11	0.08	0.10	0.31	0.49	0.46	0.63
79856	0.11	0.09	0.09	0.11	0.41	0.67	0.45	0.56
96867	0.10	0.12	0.10	0.11	0.35	0.56	0.44	0.59
Media	**0.10**	**0.11**	**0.09**	**0.11**	**0.36**	**0.56**	**0.45**	**0.60**

5.3.1.2. Esquema proporcional

Se han realizado 4 pruebas distintas para el esquema proporcional variando el valor de α y así poder ver cómo varían los resultados obtenidos dependiendo de la velocidad de enfriamiento.

Comparativa ILS-ES Esquema proporcional 10% restricciones												
	Iris			Rand			Newthyroid			Ecoli		
α	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea
0.90	0.669	0.669	0	0.755	0.725	4.2	14.101	13.874	6.2	24.815	22.282	94.4
0.92	0.679	0.675	0.6	0.742	0.724	2.4	14.054	13.835	6	25.273	21.630	135.8
0.95	0.682	0.669	2.0	0.730	0.721	1.2	14.084	13.879	5.6	25.101	24.412	100.2
0.97	0.669	0.669	0.0	0.746	0.725	2.8	14.148	13.241	24.8	25.869	22.252	134.8

Comparativa ILS-ES Esquema proporcional 20% restricciones												
	Iris			Rand			Newthyroid			Ecoli		
α	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea
0.90	0.724	0.682	13.2	0.730	0.721	2.6	14.289	14.106	10	24.844	22.118	203.2
0.92	0.681	0.674	2.2	0.716	0.716	0	14.288	14.197	5	25.740	22.569	236
0.95	0.694	0.671	7.2	0.732	0.720	3.2	14.289	14.139	8.2	25.404	22.828	192.0
0.97	0.680	0.669	3.4	0.728	0.720	2.2	14.288	14.197	5	25.742	22.490	242.4

Tiempos ILS-ES Esquema proporcional (segundos)								
	Iris		Rand		Newthyroid		Ecoli	
α	10%	20%	10%	20%	10%	20%	10%	20%
0.90	0.18	0.24	0.17	0.24	0.84	1.19	9.26	13.02
0.92	0.17	0.26	0.18	0.25	0.86	1.28	9.61	13.29
0.95	0.19	0.27	0.21	0.28	0.96	1.55	10.33	15.44
0.97	0.22	0.32	0.22	0.31	1.16	1.74	11.02	15.65

Los resultados son similares para todos los valores de α aunque, para el dataset ecoli, que podriamos considerar el más complicado de los cuatro, vemos que α = 0.9 obtiene los mejores resultados, es decir, el que enfría más rápido de todos, aun así, los resultados no son especialmente buenos.

5.4. Resultados generales

Resultados globales en el PAR con 10% restricciones												
	Iris			Rand			Newthyroid			Ecoli		
Algoritmo	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea
Greedy	1.476	0.857	97.6	2.451	1.485	134	X	X	X	51.905	39.724	454
BL	0.669	0.669	0	0.716	0.716	0	X	X	X	23.929	21.836	78
ES	0.669	0.669	0	0.716	0.716	0	14.251	13.227	28	23.968	21.360	97.2
BMB	0.669	0.669	0	0.716	0.716	0	14.054	13.835	6	26.373	22.649	138.8
ILS-BL	0.669	0.669	0	0.716	0.716	0	14.258	12.063	60	23.566	21.57	74.4
ILS-ES Cauchy	0.669	0.669	0	0.746	0.722	3.4	1.256	13.371	24.2	61.269	35.475	961.4
ILS-ES Prop 0.90	0.669	0.669	0	0.755	0.725	4.2	14.101	13.874	6.2	24.815	22.282	94.4
ILS-ES Prop 0.92	0.679	0.675	0.6	0.742	0.724	2.4	14.054	13.835	6	25.273	21.630	135.8
ILS-ES Prop 0.95	0.682	0.669	2.0	0.730	0.721	1.2	14.084	13.879	5.6	25.101	24.412	100.2
ILS-ES Prop 0.97	0.669	0.669	0	0.746	0.725	2.8	14.148	13.241	24.8	25.869	22.252	134.8

Resultados globales en el PAR con 20% restricciones												
	Iris			Rand			Newthyroid			Ecoli		
Algoritmo	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea	f	tasa_C	infea
Greedy	0.842	0.710	41.4	1.163	0.864	82.8	X	X	X	45.667	38.305	549
BL	0.669	0.669	0	0.716	0.716	0	X	X	X	24.291	22.000	171
ES	0.669	0.669	0	0.716	0.716	0	14.829	12.251	141	24.132	21.851	170
BMB	0.669	0.669	0	0.716	0.716	0	14.287	14.287	0	25.390	22.447	219.4
ILS-BL	0.669	0.669	0	0.716	0.716	0	14.443	13.606	45.8	23.638	21.849	133.4
ILS-ES Cauchy	0.704	0.676	9	0.780	0.725	6.2	14.717	12.914	98.6	60.073	35.038	1866.2
ILS-ES Prop 0.90	0.724	0.682	13.2	0.730	0.721	2.6	14.289	14.106	10	24.844	22.118	203.2
ILS-ES Prop 0.92	0.681	0.674	2.2	0.716	0.716	0	14.288	14.197	5	25.740	22.569	236
ILS-ES Prop 0.95	0.694	0.671	7.2	0.732	0.720	3.2	14.289	14.139	8.2	25.404	22.828	192.0
ILS-ES Prop 0.97	0.680	0.669	3.4	0.728	0.720	2.2	14.288	14.197	5	25.742	22.490	242.4

Lo esperable al comparar éstos algoritmos sería que los de trayectorias simples fuesen peores que los compuestos pero podemos ver que no es el caso, sólo el algoritmo ILS-BL es mejor que el algoritmo de trayectorias simples que utiliza.

Si comparamos los resultados de la BL y la BMB vemos que la segunda obtiene los peores resultados. Ésto se debe a que la búsqueda local se ejecutó con un máximo de 100000 evaluaciones mientras que, en la BMB, el límite son 10000 evaluaciones. Es obvio que el segundo va a obtener peores resultados ya que no le damos tiempo a converger. No ocurre lo mismo con el algorimo ILS-BL ya que éste trabaja sobre la misma solución que muta y mejora en cada iteración aunque el máximo de evaluaciones también sea 10000.

Ya se ha comentado la razón por la que el algoritmo ILS-ES utilizando el esquema de Cauchy obtiene resultados malos, no obstante, el esquema proporcional tampoco supera al enfriamiento simulado simple. En vista de los resultados, no parece una buena idea hibridar ILS con ES, al menos con los

parámetros utilizados, habría que realizar más pruebas para encontrar los valores de éstos que obtienen los mejores resultados posibles.

Tiempos globales en el PAR (segundos)								
	Iris		Rand		Newthyroid		Ecoli	
Algoritmo	10%	20%	10%	20%	10%	20%	10%	20%
Greedy	0.12	0.15	0.13	0.15	X	X	1.08	1.67
BL	3.81	3.52	3.54	3.73	X	X	164.24	150.95
ES	0.15	0.20	0.13	0.15	0.28	1.01	2.78	5.53
BMB	0.29	0.40	0.30	0.39	1.16	1.73	10.79	15.75
ILS-BL	0.17	0.25	0.18	0.25	0.71	1.15	10.83	15.89
ILS-ES Cauchy	0.1	0.11	0.09	0.11	0.36	0.56	0.45	0.6
ILS-ES Prop 0.90	0.18	0.24	0.17	0.24	0.84	1.19	9.26	13.02
ILS-ES Prop 0.92	0.17	0.26	0.18	0.25	0.86	1.28	9.61	13.29
ILS-ES Prop 0.95	0.19	0.27	0.21	0.28	0.96	1.55	10.33	15.44
ILS-ES Prop 0.97	0.22	0.32	0.22	0.31	1.16	1.74	11.02	15.65