UNIVERSIDAD
DE GRANADA

METAHEURÍSTICAS

GRADO EN INGENIERÍA INFORMÁTICA

PRÁCTICA 3

BÚSQUEDAS POR TRAYECTORIAS PARA EL PROBLEMA DEL AGRUPAMIENTO CON RESTRICCIONES (PAR)

Autor
Fernando Vallecillos Ruiz

DNI
77558520J

E-Mail
nandovallec@correo.ugr.es

Grupo de prácticas
MH1 Miércoles 17:30-19:30

Rama
Computación y Sistemas Inteligentes

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN

CURSO 2019-2020

Índice

1. Descripción del problema

El problema que vamos a analizar en esta práctica se trata del Agrupamiento con Restricciones (PAR). Este es una variación del problema de agrupamiento o *clustering*, el cual persigue la clasificación de objetos de acuerdo a posibles similitudes entre ellos. De esta forma, el agrupamiento es una técnica de aprendizaje no supervisado ya que permite descubrir grupos (no conocidos) en un conjunto de datos y agrupar los datos similares. Cabe destacar que no es un algoritmo en específico, sino un problema pendiente de solución. Existen multitud de algoritmos que resuelven este problema los cuales difieren en su definición de *cluster* y su método de búsqueda.

El problema de Agrupamiento con Restricciones es una generalización del problema de agrupamiento. Incorpora al proceso nueva información: las restricciones. Esto provoca un cambio en el tipo de tarea que pasa a ser semi-supervisada. Intentaremos encontrar una partición $C = \{c_1, c_2, \ldots, c_k\}$ del conjunto de datos X con n instancias que minimice la desviación general y cumpla con las restricciones en el conjunto R.

1.1. Variantes del problema

Variantes según tipo de restricciones[1]:

- *Cluster-level constraints*: se definen requerimientos específicos a nivel de *clusters* como:
  - Número mínimo/máximo de elementos
  - Distancia mínima/máxima de elementos
- *Instance-level constraints*: se definen propiedades entre pares de objetos tales como la pertenencia o no de elementos al mismo cluster

Variantes según la interpretación de las restricciones:

- Basados en métricas(*metric-based*): El algoritmo de *clustering* utiliza una distancia métrica. Esta métrica sera diseñada para que cumpla con todas las restricciones.
- Basados en restricciones(*constraint-based*): El algoritmo es modificados de manera que las restricciones se utilizan para guiar el algoritmo hacia una partición C más apropiada. Se lleva a cabo mediante la modificación de la función objetivo del *clustering*.

1.2. Formalización del problema

El conjunto de datos es una matriz X de $n \times d$ valores reales. El conjunto de datos esta formado por n instancias en un espacio de d dimensiones notadas como:

$$\vec{x_i} = \{x_{[i,1]}, \cdots, x_{[i,d]}\}$$

Un *cluster* c_i consiste en un subconjunto de instancias de X. Cada *cluster* c_i tiene asociada una etiqueta (nombre de *cluster*) l_i.

Para cada *cluster* c_i se puede calcular su centroide asociado:

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

A partir de estos, se puede calcular la distancia media intra-cluster:

$$\overline{c_i} = \frac{1}{|c_i|} \sum_{\vec{x_j} \in c_i} \|\vec{x_j} - \vec{\mu_i}\|$$

Para calcular la desviación general de la partición C podemos:

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

Dada una partición C y el conjunto de restricciones, definimos *infeasibility* como el número de restricciones que C incumple. Definimos $V(\vec{x_i}, \vec{x_j})$ como la función que, dada una pareja de instancias, devuelve 1 si viola una restricción y 0 en otro caso:

$$infeasibility = \sum_{i=0}^{n} \sum_{j=0}^{n} V(\vec{x_i}, \vec{x_j})$$

Entonces, podemos formularlo como:

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

donde λ es un parámetro de escalado para dar relevancia a *infeasibility*. Si se establece correctamente, el algoritmo optimiza simultáneamente el número de restricciones incumplidas y la desviación general.

2. Descripción de los algoritmos

2.1. Consideraciones previas

Antes de realizar una descripción formal de cada algoritmo, necesitamos describir aspectos comunes entre estos: esquema de representación de soluciones, pseudocódigo de operadores comunes o la función objetivo.

Comenzaremos hablando de las estructuras de datos utilizadas. Se ha utilizado una matriz X de $n \times d$ para guardar los datos. Se ha querido aprovechar al máximo la capacidad de paralelización de operaciones, por ello se optó por añadir las columnas necesarias a X para concentrar en una sola estructura todos los datos necesarios para calcular nuestros parámetros. Por ello, en cada algoritmo se añadirán columnas diferentes dependiendo de las necesidades de este. No obstante, existen dos columnas añadidas comunes a cualquier algoritmo. Estas son la columna *cluster*, la cual representa a que *cluster* pertenece la instancia. Y la columna *Distance to Cluster (DC)* que representa la distancia de la instancia al centroide de su cluster. De aquí en adelante, serán referidas como $x_{cluster}$ y x_{DC} respectivamente.

2.1.1. Procedimientos comunes para cálculos con individuos

Para los centroides se ha escogido una matriz M de $k \times d$ siendo k el número de *clusters*. De esta forma, para cualquier instancia x, $x_{cluster}$ indicara el índice en la matriz M. Por ello, M puede ser referido como vector de centroides. Para calcularlos, se han calculado los vectores promedios de las instancias.

Algorithm 1: Calcular centroides

```
1 CalcularCentroides (X)
      input : Matriz de datos X
      output: Vector de centroides M
2     foreach x ∈ X do
3         sum_{l_{x_i}} ← sum_{l_{x_i}} + x
4         count_{l_{x_i}} ← count_{l_{x_i}} + 1
5     centroids = sum/count
6     return centroids
```

También se ha optado por representar el conjunto R de restricciones en una matriz de $n \times n$. Vamos a contar con solo 2 tipos de restricciones *Must-Link (ML)* y *Cannot-Link (CL)*. Dada una pareja de instancias, se establece una restricción *ML*

si deben pertenecer al mismo *cluster* y de tipo *CL* si no pueden pertenecer al mismo *cluster*. Se representa una restricción *ML* entre las instancias x_i e x_j si $R[i][j] == 1$. Por otra parte, se representa una restricción *CL* entre las instancias x_i e x_j si $R[i][j] == -1$. Como se puede ver, la operación para calcular *infeasibility* dado una matriz de datos X es trivial pero explicaremos brevemente su implementación y modularización.

Para calcular *infeasibility* dado una matriz de datos X, podríamos mirarlo como la suma de *infeasibility* de cada instancia $x \in X$.

Algorithm 2: Calcular Infeasibility

```
1  CalcularInfeasibility (X)
       input  : Matriz de datos X
       output: Valor de infeasibility
2      total ← 0
3      for i ← 0 to N − 1 do
4          total ← total + CalcularInfeasibilityRowPartial(X, i)
5      return total
```

Se necesita entonces, calcular el valor de *infeasibility* para cualquier $x \in X$. Paro ello, se recorren las restricciones asociadas a la instancia seleccionada. Se descarta la restricción de una instancia consigo misma. Si la restricción es de tipo *ML* o *CL* se comprueba si se cumple o no, acumulando puntos si fuese necesario.

Algorithm 3: Calcular Infeasibility Row

```
1  CalcularInfeasibilityRow (X, r)
       input  : Matriz de datos X, Índice r
       output: Valor de infeasibility asociado a x_r
2      total ← 0
3      for i ← 0 to N − 1 do
4          if R_r[i] == 1 then
5              if X[i]_cluster ≠ X[r]_cluster then
6                  total ← total + 1
7          else if R_r[i] == −1 then
8              if X[i]_cluster == X[r]_cluster then
9                  total ← total + 1
10     return total
```

Sin embargo, si se llamase a esta función para calcular *infeasibility* de toda la matriz X, se acabaría con el doble de puntuación. Este problema se puede

solucionar de forma fácil devolviendo solo la mitad del valor acumulado. Subyace un problema de eficiencia, al ser una matriz, se calculan todas las inversas de las restricciones. Por ello, se puede crear una función para calcular de forma parcial el valor de una fila. Esta función sera llamada solo cuando queramos calcular *infeasibility* de todo el conjunto.

Algorithm 4: Calcular Infeasibility Row Partial

```
1  CalcularInfeasibilityRowPartial (X, r)
       input  : Matriz de datos X, Índice r
       output: Valor de infeasibility asociado a x_r
2      total ← 0
3      for i ← r + 1 to N − 1 do
4          if R_r[i] == 1 then
5              if X[i]_cluster ≠ X[r]_cluster then
6                  total ← total + 1
7          else if R_r[i] == −1 then
8              if X[i]_cluster == X[r]_cluster then
9                  total ← total + 1
10     return total
```

Se describen a continuación las funciones necesarias para calcular los parámetros de la función objetivo. Como se ha descrito antes, esta tendrá la siguiente forma:

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

El valor de λ se ha calculado como el cociente entre la mayor distancia entre dos instancias del conjunto y el número de restricciones:

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

El último parámetro es $\overline{C}$. Para calcularlo, primero se asignará la distancia desde cada instancia $x \in X$ a su *cluster*. Para ello, simplemente se calcula la

distancia euclídea entre ambos puntos y se asigna a su columna correspondiente.

Algorithm 5: Calcular Distancia a Centroides

```
1 CalcularDistanceCluster (X, M)
      input  : Matriz de datos X, Vector de Centroides M
      output: Matriz de datos X
2     foreach x ∈ X do
3         a ← x
4         b ← M [x_cluster]
5         x_DC ← sqrt(Σ_{i=1}^{d} (a_i − b_i)^2)
6     return X
```

Con las distancias a mano, se puede calcular cada $\overline{c_i}$ y por tanto $\overline{C}$ como la media de estos.

Algorithm 6: Calcular Desviación General

```
1 CalcularDesviacion (X)
      input  : Matriz de datos X
      output: Valor de Desviación General average
2     sum_{l_{x_i}} ← [k]
3     count_{l_{x_i}} ← [k]
4     foreach x ∈ X do
5         sum_{l_{x_i}} ← sum_{l_{x_i}} + x_DC
6         count_{l_{x_i}} ← count_{l_{x_i}} + 1
7     average = mean(sum/count)
8     return average
```

Como se puede ver, los dos últimos algoritmos están muy entrelazados entre si. Han sido escritos de forma separada para comprender su comportamiento de una forma fácil y simple. Sin embargo, han sido implementado de forma conjunta para mejorar la eficiencia temporal.

2.1.2. Procedimientos comunes para trayectorias

Se comienza a describir procedimientos comunes en los algoritmos basados en trayectorias. En todos los algoritmos se comienza preparando todas las variables

y estructuras de datos necesarias para realizar la búsqueda:

Algorithm 7: Preparación Búsqueda Local

input :
output:
1 $N \leftarrow GenerateVirtualNeighbourhood(X)$
2 $N \leftarrow Shuffle(N)$
3 $X \leftarrow GenerateRandomSolution(X)$
4 **if** *SolutionNotValid(X)* **then**
5 Exit()
6 $M \leftarrow CalcularCentroides(X)$
7 $X, SumDist, ClusterCount \leftarrow CalculateDistanceSeparate(X, M)$
8 $nEvaluations \leftarrow 0$
9 $TotaInfeasibility \leftarrow CalcularInfeasibility(X)$
10 $ObjectiveValue =$
$CalcularObjectiveValue(SumDist, ClusterCount, Lambda, TotalInfeasibility)$
11 $repeated \leftarrow False$
12 $SumValuesCluster \leftarrow SumInstances(X)$

Se empieza creando el vecindario virtual de $n \times k$ elementos como explicado anteriormente y se baraja para aleatorizar los accesos. Se crea una solución asignando un *cluster* aleatorio a cada instancia, y se comprueba que sea válida (ningún *cluster* vacío). Usamos entonces la función $CalculateDistanceSeparate$. Se explicó anteriormente en los algoritmos 5 y 6 cómo calcular las distancias a centroides y la desviación general. Se mencionó que las funciones se podían combinar en una sola. Lo cual devolvería X y *average*. Sin embargo, podemos no realizar el último paso y devolver *sum* y *count*. Siendo *sum* un vector con la suma de distancias por cada *cluster*. Y *count*, un simple conteo de cuantas instancias pertenecen a cada *cluster*.

Siguiendo con la función, creamos una variable para contar el número de evaluaciones y calculamos *infeasibility*. Se llama a una función para calcular el valor de la función objetivo con la fórmula ya explicada. Se crea la variable $repeated$, la cuál se utiliza como criterio de parada (solamente en algunos de los algoritmos). Cambiará de valor solo cuando hayamos explorado todo el vecindario virtual y no se haya realizado ningún cambio. Por último, se crea la función $SumInstances$ la cual nos servirá para factorizar el cálculo de los centroides. Simplemente devuelve un vector de tamaño k con la suma de las instancias asignadas a cada *cluster*

Se discute brevemente las variables y su factorización. La factorización de *centroides* se ha conseguido gracias a las dos variables $SumValuesCluster$ y $ClusterCount$. Como se vio en el algoritmo 1, calculamos los centroides hallando los vectores promedios. Si se para un paso antes las variables *sum* y *count* pueden

mantenerse separadas y ser actualizadas de forma simple. Al añadir/quitar una instancia de un determinado *cluster*, solo se necesita sumar/restar respectivamente de los valores acumulados.
La factorización de la desviación general se ha conseguido gracias a las variables $SumDist$ y $ClusterCount$. Como se vio en el algoritmo 6, la desviación general puede hallarse como la media de las desviaciones *intra-cluster*. A su vez, las desviaciones *intra-cluster* pueden hallarse como la media de las distancias en un *cluster*. Se puede mantener acumuladas la suma de las distancias *intra-cluster* en $SumDist$ y actualizar de manera individual los valores.

Uno de los aspectos comunes más importantes es el operador de vecino. Para aumentar la eficiencia en tiempo de nuestro algoritmo crearemos un vecindario virtual compuesto de parejas *índice, valor*. Es decir, cada una de estas parejas representará una solución (factible o no) resultado de asignar a la instancia x_indice el *cluster* con índice *valor*. Este vecindario virtual tendrá $n \times k$ componentes. Cabe destacar que al menos n componentes no tendrán utilidad, ya que asignan a una instancia el *cluster* que ya tiene asignado. Además se puede dar el caso de que una asignación deje un *cluster* vacío. Está solución no es factible y por lo tanto será descartada.
Ya que estas comprobaciones son de carácter trivial, se prefiere comprobar las parejas antes de usarlas que generar este vecindario virtual con solo parejas factibles. Estas parejas serán creadas y barajadas, posteriormente se accederán a ellas a través de la función *GetNeighbour*. Esta simplemente devuelve el primer vecino y rota el vector. De esta forma, el vecino devuelto puede ser devuelto de nuevo si y solo si, se han devuelto todos los otros vecinos una vez. Esto permite iterar sobre el vector de forma indefinida y conocer si se han intentado todos los posibles vecinos de una misma solución. Además, permite barajarlo para mantener la aleatoriedad entre las iteraciones.

Algorithm 8: Get Neighbour

```
1 GetNeighbour (N)
      input  : Vector de Parejas N
      output: Vector de Parejas N, Pareja n
2     n ← N_0
3     N ← RotarVector(N)
4     return N, n
```

Se describe entonces otro pequeño módulo sobre como factorizar el cálculo de *infeasibility* y el conteo de los *cluster*. Este módulo cambiará la asignación de una instancia $x \in X$ desde el *cluster* $OldCluster$ a $NewCluster$. Gracias al uso de la función $CalcularInfeasibilityRow$ se puede calcular la contribución en *infeasibility* de una instancia concreta. Se sustrae del valor de *infeasibility* total. Y se actualiza el conteo en $ClusterCount$ para reflejar que una instancia menos

pertenece a ese *cluster*. Se cambia la asignación de x al nuevo *cluster* y se vuelven a actualizar las variables de la misma forma.

Algorithm 9: Actualizar Count y Infeasibility

```
1 UpdateCountInfeasibility
    input  : TotalInfeasibility
    output: TotalInfeasibility
2   TotalInfeasibility ←
      TotalInfeasibility − CalcularInfeasibilityRow(X, Index)
3   ClusterCount_OldCluster ← ClusterCount_OldCluster − 1
4   ChangeCluster(X, Index, NewCluster)
5   TotalInfeasibility ←
      TotalInfeasibility + CalcularInfeasibilityRow(X, Index)
6   ClusterCount_NewCluster ← ClusterCount_NewCluster + 1
7   return TotalInfeasibility
```

Por último, se necesita explicar la función $RestoreValues()$. El cálculo de las nuevas soluciones se realiza a partir de vecinos y sustituye a la solución actual. Esta función permite volver a la anterior solución actual. Gracias a la factorización de todas las variables esto se puede hacer de forma rápida revirtiendo las operaciones. Es decir, se restauran los valores de:

- El *cluster* asignado a la instancia $x_{Cluster}$
- El valor de *infeasibility*
- El valor de $ClusterCount$
- El valor de $SumDist$ gracias a $OldDist$
- El valor de los centroides M y la suma de valores $SumValuesCluster$

2.2. Algoritmos de comparación: COPKM

Para comparar nuestras metaheurísticas, se ha decidido usar el algoritmo *COP-KMeans* introducido por Wagstaff et al.[2] con una interpretación débil de las restricciones. Se comenzará explicando cada una de las funciones necesarias para la implementación, terminando con el pseudocódigo completo.

Primero, se explica como obtener los primeros centroides. Existen diferentes heurísticas solo para esta tarea pero se ha decidido crearlos de forma aleatoria. Para cada característica en la matriz de datos, se calcula el mínimo y máximo. De esta forma, podemos obtener centroides dentro del espacio de características. Sin embargo, pueden haber *outliers* en los datos que empeoren de manera significativa esta técnica, así que se opta por dividir el valor *random* para centrarlo de forma suave.

Algorithm 10: Generar Centroides Aleatorios

```
1 GenerateRandomCentroids (X)
    input  : Matriz de datos X
    output: Vector de centroides M
2   M ← [ ][ ]
3   for h ← 0 to d do
4       min ← MinValue(X_h)
5       max ← MaxValue(X_h)
6       for i ← 0 to k do
7           M ← RandomBetween(min, max)/1.5
8   return M
```

Teniendo ya unos centroides calculados, se puede pasar a la primera asignación de la matriz de datos. Aunque el pseudocódigo de la función parezca largo, ha sido diseñado de una forma simple y eficiente para tener en cuenta las situaciones

extremas posibles. Se irá descomponiendo y explicando en varias partes.

Algorithm 11: Inicialización Matriz

```
1  InicializarMatriz (X, M)
       input  : Matriz de datos X, Vector de centroides M
       output: Matriz de datos X actualizada
2      ClusterCount[k] ← 0
3      for w ← 0 to n − 1 do
4          i ← randomIndex[w]
5          x ← X_i
6          for CIndex ← 0 to k do
7              x_Cluster = CIndex
8              x_InfCIndex = CalcularInfeasibilityRowFirst(X, i)
9          own ← LeastInfeasibilityClusterOf(x)
10         x_cluster ← own
11         ClusterCount_own ← ClusterCount_own + 1
12         for CIndex ← 0 to k do
13             if CIndex ≠ own & & x_InfOwn == x_InfCIndex then
14                 ownDistance ← Distancia(x, M_own)
15                 otherDistance ← Distancia(x, M_CIndex)
16                 if otherDistance > ownDistance then
17                     ClusterCount_own ← ClusterCount_own − 1
18                     x_cluster ← CIndex
19                     own ← CIndex
20                     ClusterCount_own ← ClusterCount_own + 1
21     for i ← 0 to k do
22         if ClusterCount_i == 0 then
23             w ← i
24             while CountCluster_{X_{w_Cluster}} ≥ k & & w < n − 1 do
25                 w ← w + 1
26             X_{w_Cluster} ← i
27     return X
```

Primeramente, se define un bucle para iterar sobre los datos según un índice aleatorio. Por cada instancia x obtenida, le asignaremos en su columna correspondiente un *cluster*. Se calcula *infeasibility* suponiendo x pertenece a cada uno de los *clusters* y se guarda el resultado en una columna. Es decir, se han creado k columnas adicionales que guardan el valor $infeasibility$ con respecto a cada uno de los k *clusters*.

Esto sería suficiente para una gran parte de los casos. Se ha querido aumentar la seguridad de la heurística. Por ello, con el segundo bucle se comprueba si existe otro *cluster* con el mismo valor *infeasibility*. Si es así, se elige aquel con menor distancia al centroide respectivo.

Se ha querido asegurar otro caso extremo. Puede darse el caso de que algún *cluster* sea vacío. Por ello, nos aseguramos a partir de la linea **21**. En cada instancia, se ha llevado la cuenta a donde se asignaba en *ClusterCount*. Por ello, se puede asegurar que ningún *cluster* esté vacío. Si lo estuviera, se ha buscado la siguiente instancia cuyo *cluster* tenga k elementos o más y se ha cambiado su pertenencia al *cluster* vacío. Se realiza una búsqueda de esta forma para evitar que el cambio deje a este segundo *cluster* vacío (incluso cuando varios *cluster* tomen de él).

Se podría clarificar que la función *CalcularInfeasibilityRowFirst* es muy similar a la función explicada anteriormente *CalcularInfeasibilityRow*. Con una pequeña diferencia, comprueba si la segunda instancia x_i(a la que comparamos), ha sido asignada o no. Si no, no comprueba las restricciones pues no tiene nada con lo que comparar.

Por último, se explicará la función de asignación regular. Será la usada en bucle en nuestra implementación y es muy similar a la inicialización de matriz

previamente vista.

Algorithm 12: Asignación Regular

```
1  RegularAssignation (X, M)
       input  : Matriz de datos X, Vector de centroides M
       output: Matriz de datos X actualizada
2      ClusterCount ← CountClusters(X)
3      for w ← 0 to n − 1 do
4          i ← randomIndex[w]
5          x ← X_i
6          own ← x_cluster
7          if ClusterCount_own == 1 then
8              NextIterationFor()
9          ClusterCount_own ← ClusterCount_own − 1
10         for CIndex ← 0 to k do
11             x_Cluster = CIndex
12             x_InfCIndex = CalcularInfeasibilityRow(X, i)
13         own ← LeastInfeasibilityClusterOf(x)
14         x_cluster ← own
15         ClusterCount_own ← ClusterCount_own + 1
16         for CIndex ← 0 to k do
17             if CIndex ≠ own & & x_InfOwn == x_InfCIndex then
18                 ownDistance ← Distancia(x, M_own)
19                 otherDistance ← Distancia(x, M_CIndex)
20                 if otherDistance > ownDistance then
21                     ClusterCount_own ← ClusterCount_own − 1
22                     x_cluster ← CIndex
23                     own ← CIndex
24                     ClusterCount_own ← ClusterCount_own + 1
25     return X
```

Se puede apreciar que la estructura es similar a la inicialización de la matriz. Se explicará la función basándose en las diferencias. Se ha necesitado contar los el número de instancias de cada *cluster* al comienzo de la función. Se ha realizado este cambio ya que se necesita comprobar que no se deje ningún *cluster* vacío. Si se verifica que tiene más de un elemento, podemos comprobar los puntos *infeasibility* con respecto a otros *cluster*. El resto es igual que el previamente explicado.

Con todo lo necesario ya explicado, se pasa al pseudocódigo del programa principal. Toda la implementación se centra en torno a un bucle *while* principal. Primero se inicializa un índice aleatorio de tamaño n, el cual servirá para aleatori-

zar los acceso a X. Como se ha explicado, se inicializan el vector M de centroides y la estructura de datos X. Creamos un vector $OldClusters$. El bucle consiste de tres sencillos pasos:

1. Guardamos la columna con la asignación de *clusters* en nuestra variable $OldClusters$.
2. Calculamos los nuevos centroides a partir de la asignación de *clusters* actual.
3. Realizamos la nueva asignación de *clusters* a la instancias.

Como condición del bucle, se asegura que haya habido algún cambio en la asignación de *clusters* a las instancias.

Algorithm 13: COP-KMeans Débil

```
1  COPKmeansSoft (X)
       input  : Matriz de datos X, Matriz de restricciones R
       output: Matriz de datos X con asignaciones a clusters
2      randomIndex ← RandomShuffle(n)
3      M ← GenerateRandomCentroids(X)
4      X ← FirstAssignation(X, M)
5      OldClusters ← [ ]
6      while OldClusters ≠ X_Clusters do
7          OldClusters ← X_Clusters
8          M ← CalcularCentroides(X)
9          X ← RegularAssignation(X, M)
10     return X
```

Con esto, se acaba la explicación de nuestro de algoritmo de comparación. Como se ha visto, este algoritmo prioriza siempre *infeasibility* sobre la distancia (aunque la toma en cuenta en casos de empate). También es un algoritmo que luce por su simpleza y velocidad con la que alcanza soluciones factibles.

2.3. Algoritmo de Búsqueda Local

A continuación se describe el algoritmo de búsqueda local considerando el esquema del primer mejor. También se realiza una interpretación débil de las restricciones. Se han descrito anteriormente la mayoría de funciones y componentes de esta búsqueda. Sin embargo, se explicarán algunos componentes más y se describirá las modificaciones realizadas a las funciones así como un esquema general de la implementación.

Debido a las optimizaciones realizadas en el código, es difícil marcar separaciones exactas sobre los módulos. Se ha realizado una factorización de las variables para ahorrar la máxima cantidad de operaciones posibles y evitar cualquier operación redundante. Se intentará dividir el pseudocódigo en lo posible y se explicará de forma gradual. Por aumentar la claridad de la explicación se han sacado pequeños módulos de código de la función principal. Estos se verán identificados por ausencia de *input* o *output*. En la implementación, se encontrarían dentro del código de la función principal.

Con la mayoría de los componentes descritos, se comienza a describir el algoritmo final. Se muestra primero el pseudocódigo y luego se comenzará explicar por

partes:

Algorithm 14: Búsqueda Local Débil

```
1  LocalSearchSoft (X)
       input  : Matriz de datos X, Matriz de restricciones R
       output: Matriz de datos X con asignaciones a clusters
2      PrepareLocalSearch(···)
3      while nEvaluations < 100000 & & !repeated do
4          FirstIteration ← True
5          FirstNeighbour ← N_0
6          OldObjectiveValue ← ObjectiveValue
7          OldInfeasibility ← TotalInfeasibility
8          while OldObjectiveValue ≤ ObjectiveValue & &
            nEvaluations < 100000 do
9              nEvaluations ← nEvaluations + 1
10             N, Neighbour ← GetNeighbour(N)
11             if Neighbour == FirstNeighbour & & !FirstIteration then
12                 repeated ← True
13                 Break
14             FirstIteration ← False
15             OldCluster ← GetCluster(X, Neighbour)
16             Index ← Neighbour_Index
17             NewCluster ← Neighbour_Value
18             if ClusterCount_OldCluster == 1 || OldCluster == NewCluster
                then
19                 Continue
20             M, SumValuesCluster ← CalculateCentroidesOptimizado(···)
21             X, SumDist, OldDist ← CalcularDistanciasOptimizado(···)
22             ObjectiveValue = CalcularObjectiveValue(···)
23             if OldObjectiveValue ≤ ObjectiveValue then
24                 RestoreValues(OldDist, ···)
25     return X
```

Se comienza realizando todas las preparaciones necesarias de variables explicadas anteriormente. Se entra al bucle principal de la función. Este bucle *while* continuará mientras se hayan realizado menos de 100.000 evaluaciones o no se hayan explorado todos los vecinos de una solución. Las variables $FirstIteration$ y $FirstNeighbour$ son variables auxiliares usadas en la condición de parada. Como se ha dicho, ayudarán a parar el algoritmo si se ha explorado todo el vecindario. En las siguientes lineas se guardan los valores de *infeasibility* y la función objetivo para poder restaurarlos luego. Se entra entonces en el segundo bucle de la función.

Este bucle se utiliza para explorar todos los vecinos de la solución actual hasta que encontrar una mejor solución. Este bucle se parará al encontrar una mejor solución a la actual o al pasar las 100.000 evaluaciones. Al principio se aumentan el número de evaluaciones. Entonces se obtiene el primer vecino de la lista a partir de la función $GetNeighbour$. Se realiza una comprobación para ver que este vecino no ha sido usado anteriormente. Si lo ha sido, significa que se han probado todos los otros posibles vecinos y por lo tanto el programa termina. Se usan $OldCluster$, $NewCluster$ e $Index$ como variables auxiliares para acceder cómodamente a los valores del vecino y el mantener la antigua asignación de *cluster*. Se realiza una última comprobación para asegurar que el vecino obtenido será una solución válida. Es decir, no deja ningún *cluster* vacío y no nos devolverá la misma solución. Si lo hace, salta al siguiente vecino.

Las funciones $CalcularCentroidesOptimizado$ y $CalcularDistanciasOptimizado$ utilizan lo explicado anteriormente para calcular los centroides y la desviación general de la manera más eficiente posible. Con todos los parámetros ya calculados se puede llamar a la función $CalcularObjectiveValue$ y actualizar el valor de la función objetivo. Se realiza entonces una comparación con el valor de la función objetivo anterior. Si la solución es mejor, todos los valores han sido ya actualizados y se saldrá del bucle con la nueva solución. Si la solución es peor, se restauran las diferentes variables a como estaban al principio del bucle. Es decir, se restauran los valores de:

- El *cluster* asignado a la instancia $x_{Cluster}$
- El valor de *infeasibility*
- El valor de $ClusterCount$
- El valor de $SumDist$ gracias a $OldDist$
- El valor de los centroides M y la suma de valores $SumValuesCluster$

Con esto se termina de describir el algoritmo de búsqueda local. Este algoritmo ciclará hasta haber completado todas las evaluaciones o haber explorado todos los vecinos locales de la solución actual. Se trata de un algoritmo fácil de entender pero algo más complicado de implementar. Los ajustes graduales para la factorización de los parámetros pueden ser objeto de confusiones y errores. Sin embargo, es posible alcanzar una solución buena en un tiempo corto gracias a la factorización.

3. Algoritmos de Búsqueda por Trayectorias

3.1. Algoritmo de Búsqueda Multiarranque Básica

A continuación se describe el algoritmo de Búsqueda Multiarranque Básica (BMB). También se realiza una interpretación débil de las restricciones. Se han descrito anteriormente la mayoría de funciones y componentes de esta búsqueda ya que se utiliza principalmente el algoritmo de búsqueda local descrito anteriormente.

La búsqueda multiarranque consiste en generar un número de soluciones (número arbitrario o con condiciones de parada) en la región factible y aplicar la búsqueda local a estas. Por último, se devolverá la mejor solución entre todos los resultados generados por las búsquedas locales. La búsqueda multiarranque básica se caracteriza porque las soluciones iniciales se generan de forma aleatoria.

Algorithm 15: Basic Multi-Start Local Search

```
1 BMB (X, R)
    input  : Matriz de datos X, Matriz de restricciones R
    output: Matriz de datos X con asignaciones a clusters
2   BestSolution ← []
3   for i ← 0 to it do
4       PrepareLocalSearch(···)
5       Result ← LocalSearch(···)
6       if Result < BestSolution then
7           BestSolution ← Result
8   X ← BestSolution
9   return X
```

Se crea un *array* donde guardar los parámetros necesarios para una solución. Se entra entonces en un bucle donde se generan un total de *it* soluciones aleatorias y se les aplica una BL de forma iterativa. Si alguno de los resultados es mejor que la mejor solución guardada hasta el momento, se actualiza el valor (se compara el valor de la función objetivo). Por último, se establece la mejor solución como la elegida en el conjunto de datos y se devuelve como resultado final.

3.2. Algoritmo de Enfriamiento Simulado

Se comienza la descripción del algoritmo de Enfriamiento Simulado (ES). Este algoritmo tiene un criterio probabilístico de aceptación de soluciones basado en termodinámica. Permite que algunos movimientos sean hacia soluciones peores,

permitiendo salir de óptimos locales. Se realiza a través de una función de probabilidad que hará disminuir la probabilidad de estos movimientos hacia soluciones peores conforma avanza la búsqueda. Es decir, diversifica al principio e intensifica al final.

El criterio de aceptación de soluciones vecinas varía a lo largo de su ejecución. Hace uso de una variable llamada Temperatura T que se va reduciendo en cada iteración mediante un mecanismo de enfriamiento hasta alcanzar una Temperatura Final T_f. En cada una de las iteraciones se generan un número concreto de vecinos $L(T)$ y dependiendo del criterio se acepta o no. Esta nueva solución sustituye a la solución actual si es mejor o según una probabilidad dependiente de la temperatura y la diferencia de costes. A mayor temperatura, mayor la probabilidad de aceptación de soluciones peores que la actual. A menor diferencia de coste, mayor la probabilidad de aceptación de soluciones peores. Se irán discutiendo los valores de los parámetros usados en el algoritmo.

Es recomendable usar un valor dependiente del problema. En este caso se utiliza la siguiente formula:

$$T_0 = (\mu / -ln(\phi)) * C(S_0)$$

$$\mu = \phi = 0.3$$

Tanto por uno ϕ de probabilidad de que una solución sea un μ por uno peor que la solución inicial S_0.

Existen multitud de mecanismos de enfriamiento. Se han elegido dos de ellos:

- Descenso geométrico: $T_{k+1} = \alpha * T_k$. Siendo k la iteración actual y α una constante usualmente $\in [0.8, 0.99]$
- Cauchy Modificado: $T_{k+1} = T_k/(1 + \beta * T_k)$. Tal que con M iteraciones: $\beta = (T_0 - T_f)/(M * T_0 * T_f)$

También se necesita establecer una velocidad de enfriamiento. $L(T)$ debe ser suficientemente grande como para que el sistema llegue a alcanzar su estado estacionario para esa temperatura. Existen multitud de opciones, entre estas se encuentran:

- Máximo número de vecinos generados
- Máximo número de vecinos aceptados

Por último, se ha de establecer las condiciones de parada. En principio el algoritmo se para cuando T sobrepasa a T_f o luego de un número fijo de iteraciones. Además, se puede añadir otras condiciones como que no se haya aceptado ningún vecino de los generados en la iteración actual (algoritmo probablemente estancado). De esta forma, se crea un equilibro en la búsqueda que evita malgastar recursos.

Con todos los parámetros indicados se comienza a explicar el algoritmo:

Algorithm 16: Enfriamiento Simulado

```
1  SimAnnealing (X)
       input  : Matriz de datos X, Matriz de restricciones R
       output: Matriz de datos X con asignaciones a clusters
2      PrepareSearch(···)
3      BestSolution ← []
4      while T < T_f do
5          FirstNeighbour ← N_0
6          N ← Shuffle(N)
7          Accepted ← 0
8          Generated ← 0
9          while
            Accepted < MaxAccepted  &  &  Generated < MaxGenerated do
10             N, Neighbour ← GetNeighbour(N)
11             OldCluster ← GetCluster(X, Neighbour)
12             Index ← Neighbour_Index
13             NewCluster ← Neighbour_Value
14             if ClusterCount_OldCluster == 1 || OldCluster == NewCluster
                then
15                 Continue
16             Generated ← Generated + 1
17             M, SumValuesCluster ← CalculateCentroidesOptimizado(···)
18             X, SumDist, OldDist ← CalcularDistanciasOptimizado(···)
19             ObjectiveValue = CalcularObjectiveValue(···)
20             Δf ← ObjectiveValue − OldObjectiveValue
21             LuckInsert ← Random() < Exp(−Δf/T)
22             if Δf < 0| | LuckInsert then
23                 Accepted ← Accepted + 1
24                 UpdateBestSolution(BestSolution, X)
25             else
26                 RestoreValues(OldDist, ···)
27         T ← CoolTemperature(T)
28         if Accepted == 0 then
29             Break
30     return BestSolution
```

El algoritmo tiene una estructura similar a la búsqueda local. Se comenzará iniciando todos los variables necesarias. Se comparte el código de inicialización de la búsqueda local y se añaden algunas nuevas variables como las temperaturas o

las variables asociadas a las secuencias de enfriamiento. Se ha explicado que este algoritmo puede aceptar soluciones peores que la actual, por ello se crea la variable *BestSolution* donde guardar la mejor solución encontrada hasta el momento. Se entra entonces en el bucle principal. Se seguirá iterando mientras que la temperatura sea mayor que la temperatura final. La estructura es similar a la búsqueda local. Se genera el primer vecino y se resetean las variables usadas en el bucle interno. Este bucle genera los vecinos de forma aleatoria (descrito anteriormente). Luego de comprobar que es una solución factible se pasa a calcular el valor de la función objetivo para esta. La variable Δf permite saber si se ha alcanzado una solución mejor que la actual. Por otro lado, la variable $LuckInsert$ establece la posibilidad de que una solución peor a la actual pueda reemplazar a esta. Si se cumple cualquiera de estas opciones, la nueva solución es aceptada y, en caso de ser necesario, se actualiza la mejor solución encontrada. Si no es aceptada, se restauran los valores. Este bucle interno se ve controlado por la velocidad de enfriamiento. En este caso, se han establecido cotas superiores en el número de vecinos generado y aceptados. Al salir del bucle, se emplea el mecanismo de enfriamiento seleccionado. Además, en este caso, se ha añadido una nueva condición de parada para evitar estancamientos. Por último, se devolverá la mejor solución alcanzada en todo el recorrido (puede no ser la actual).

Con esto, se termina de describir el algoritmo de enfriamiento simulado. Se puede apreciar una estructura muy similar a la búsqueda local pero añadiendo la posibilidad de evitar óptimos locales (mayor desventaja de la BL). Es un algoritmo con cierta dificultad al introducir diferentes variables complejas y mecanismos de iteración. Aún así, permanece siendo fácil de entender y simple de implementar.

3.3. Algoritmo de Búsqueda Iterativa

El algoritmo de búsqueda iterativa (Iterative Local Search) se basa en la aplicación repetida de un algoritmo de búsqueda a una solución inicial que se obtiene por mutación de un óptimo local previamente encontrado. Esta búsqueda se puede dividr en cuatro componentes:

- Solución Inicial (usualmente aleatoria)
- Procedimiento de modificación (mutación) que aplica un cambio brusco para obtener una solución intermedia
- Procedimiento de búsqueda
- Criterio de aceptación que decide a qué solución se aplica la mutación (usualmente la mejor)

El procedimiento de modificación elegido se trata del operador de mutación por segmento. Se escoge un segmento de longitud fija de la solución y reasignar de forma aleatoria las etiquetas de las instancias asociadas a las posiciones contenidas en el segmento. De esta forma:

Algorithm 17: Mutación Fuerte

```
1 Mutation (X, length)
      input  : Array solución S, Longitud del segmento length
      output: Array solución S
2     BestSolution ← []
3     start ← RandomInRange(0, n − 1)
4     end ← (start + length) %n
5     for i ← 0 to length do
6         index ← (end + i) %n
7         S[index] ← RandomInRange(0, k − 1)
8     return S
```

Esta simple función simplemente aplica una fuerte mutación a la solución dada. La longitud determinará como de lejos se quiere la nueva solución con respecto de la solución inicial. Una longitud muy pequeña no provocaría ningún cambio, ya que la solución seguiría convergiendo al mismo óptimo local. Una longitud muy grande provoca una solución sin ninguna relación con la solución óptima, por lo que se acercaría más al comportamiento de un algoritmo BMB.

Habiendo discuto los principales componentes se pasa a describir el algoritmo de forma general:

Algorithm 18: Búsqueda iterativa

```
1 ILS (X, R)
      input  : Matriz de datos X, Matriz de restricciones R
      output: Matriz de datos X con asignaciones a clusters
2     BestSolution ← []
3     PrepareLocalSearch(···)
4     BestSolution ← Search()
5     for i ← 0 to it do
6         NewSolution ← Mutation(BestSolution)
7         PrepareSearch(···)
8         NewSolution ← Search(···)
9         UpdateBestSolution(BestSolution, NewSolution)
10    X ← BestSolution
11    return X
```

La estructura del algoritmo es muy simple y directa. Se comienza realizando una búsqueda inicial obteniendo así una primera solución. Tras esto, se entra en el bucle principal del algoritmo. Este aplica una mutación fuerte sobre la mejor solución encontrada hasta ahora. Esta mutación pretende alejar la nueva solución lo suficiente como para que no converja en el mismo óptimo local, pero manteniendo buenos cromosomas. Se aplica de nuevo la búsqueda seleccionada sobre esta solución. Si esta converge en una solución mejor o igual a la mejor encontrada hasta el momento la sustituye. Si no, será descartada.

Como se puede apreciar es un algoritmo muy sencillo y fácil con corta implementación si ya se han desarrollado los componentes anteriores. Aunque la forma básica del algoritmo use una búsqueda local, se ha querido generalizar el código para mostrar que se podría realizar con diferentes tipos de búsqueda (más simples o complejas).

4. Desarrollo de la práctica

La práctica se ha implementado en **Python3** y ha sido probada en la versión 3.6.9. Por tanto, se recomienda encarecidamente utilizar un intérprete de Python3. Se han utilizado diferentes paquetes con funciones de utilidad general: el módulo **time** para la medición de tiempos, el módulo **random** para generar números pseudoaleatorios, el módulo **scipy** para calcular λ, el módulo **math** para el cálculo de distancias euclidianas y el módulo **numpy** para gestionar matrices de forma eficiente. Adicionalmente, se han creado gráficos con *Calc* a partir de los datos obtenidos.

En cuanto a la implementación, se ha creado un programa para cada uno de los cuatro algoritmos: **bmb.py**, **annealing.py**,**ils.py** y **ils-annealing.py**. Se realizan todos las operaciones comunes y no relevantes (carga de datos, restricciones, semillas para números aleatorios) fuera del código en el cual se realizan medidas temporales. Se ha creado otro programa **comparison.py** para escribir un fichero con los datos necesarios para realizar las comparaciones.

5. Manual de usuario

Para poder ejecutar el programa, se necesita un intérprete de **Python3**, como se ha mencionado. Además, para instalar los módulos se necesita el gestor de paquetes **pip** (o **pip3**).

Para ejecutar el programa basta con ejecutar el siguiente comando:

```
$ python3 main.py
```

Los programas **bmb.py** y **ils.py** se pueden lanzar con o sin argumentos. Al lanzarse sin argumentos se realiza una ejecución por defecto. Para especificar una ejecución se lanza con los siguientes 4 argumentos:

```
$ python3 bmb.py [dataset] [Restr.%]
                           [Seed] [LambdaMod]
```

- **dataset** $\in$ { **ecoli**, **iris**, **rand**, **newthyroid**}
- **Restr. %**: nivel de restricción $\in$ {**10**, **20**}
- **Seed**: semilla $\in \mathbb{Z}$
- **LambdaMod**: modificador de $\lambda \in \mathbb{Q}$ (default = 1)

Un ejemplo en una ejecución normal sería:

```
$ python3 bmb.py ecoli 10 123 1
```

Para la ejecución de los programas **annealing.py** y **ils-annealing.py** se añaden dos argumentos a los anteriores descritos:

- **Cauchy**: establecer esquema de Cauchy como mecanismo de enfriamiento $\in$ {**si**, **no**}
- **Alfa**: establece constante para el descenso geométrico si no se ha elegido el esquema de Cauchy $\alpha \in \mathbb{Q}$

Se ejecutaría de la siguiente forma:

```
$ python3 annealing.py ecoli 10 123 1 no 0.95
```

Por otra parte, se puede comentar brevemente **comparison.py**. Será utilizado para iterar ejecuciones de los programas y guardar los resultados en archivos csv. Se ha usado tanto para la obtención de valores de las tablas, como para los otros análisis y experimentos realizados. Se pueden modificar la lista de valores con los que se quiere iterar al principio del programa de una manera sencilla y rápida.

6. Experimentación y análisis de resultados

6.1. Descripción de los casos del problema

Para analizar el rendimiento de los algoritmos, se han realizado pruebas sobre 3 conjuntos de datos:

- **Iris**: Conjunto de datos clásico en la ciencia de datos. Contiene información sobre las características de tres tipos de flor Iris. Debe ser clasificado en 3 clases.
- **Ecoli**: Conjunto de datos que contiene medidas sobre ciertas características de diferentes tipos de células que pueden ser empleadas para predecir la localización de cierta proteínas. Debe ser clasificado en 8 clases.
- **Rand**: Conjunto de datos artificial generado en base a distribuciones normales. Debe ser clasificado en 3 clases.
- **NewThyroid**: Conjunto de datos que contiene medidas cuantitativas tomadas sobre la glándula tiroides de 215 pacientes. Debe ser clasificado en 3 clases.

A cada conjunto de datos le corresponde 2 conjuntos de restricciones, correspondientes al 10 % y 20 % del total de restricciones posibles. Con esto, el total de casos de estudio principal es 8.

6.2. Análisis de los resultados

Se comienza observando los resultados generales de los casos de estudio. Como se ha descrito, se han realizado un total de 8 casos de estudio distintos. Debido a la variabilidad de resultados dependiendo del punto inicial, cada caso de estudio se ejecutará con 5 semillas distintas. Estas nos aseguran una solución inicial distinta y acceso aleatorio diferente en cada ejecución.

6.2.1. Análisis de la Búsqueda Multiarranque Básica

Como ya se ha explicado, es un algoritmo bastante simple basado en búsqueda local. Se crean un total de 10 soluciones iniciales a las que se le aplicará BL. Se mantiene la restricción de realizar un máximo de 100.000 evaluaciones de la función objetivo por lo que se han repartido estas de forma equitativa entre las

10 soluciones iniciales. Se puede esperar ver resultados similares o ligeramente mejores con respecto a la BL al converger desde diferentes puntos. Se exponen los resultados:

	Iris				**Ecoli**				**Rand**				**NewThyroid**			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	0.67	0.00	0.67	9.97	22.10	158.00	26.47	94.09	0.76	0.00	0.76	8.45	13.83	6.00	14.07	33.64
Seed 456	0.67	0.00	0.67	9.90	22.83	101.00	25.62	86.10	0.76	0.00	0.76	7.57	13.83	6.00	14.07	32.46
Seed 789	0.67	0.00	0.67	10.53	23.68	114.00	26.83	88.35	0.76	0.00	0.76	7.90	13.83	6.00	14.07	30.38
Seed 101112	0.67	0.00	0.67	10.76	22.55	155.00	26.84	85.57	0.76	0.00	0.76	8.33	13.83	6.00	14.07	30.93
Seed 131415	0.67	0.00	0.67	10.13	23.05	142.00	26.98	90.02	0.76	0.00	0.76	7.67	13.83	6.00	14.07	29.74
Media	0.67	0.00	0.67	10.26	22.84	134.00	26.55	88.83	0.76	0.00	0.76	7.98	13.83	6.00	14.07	31.43

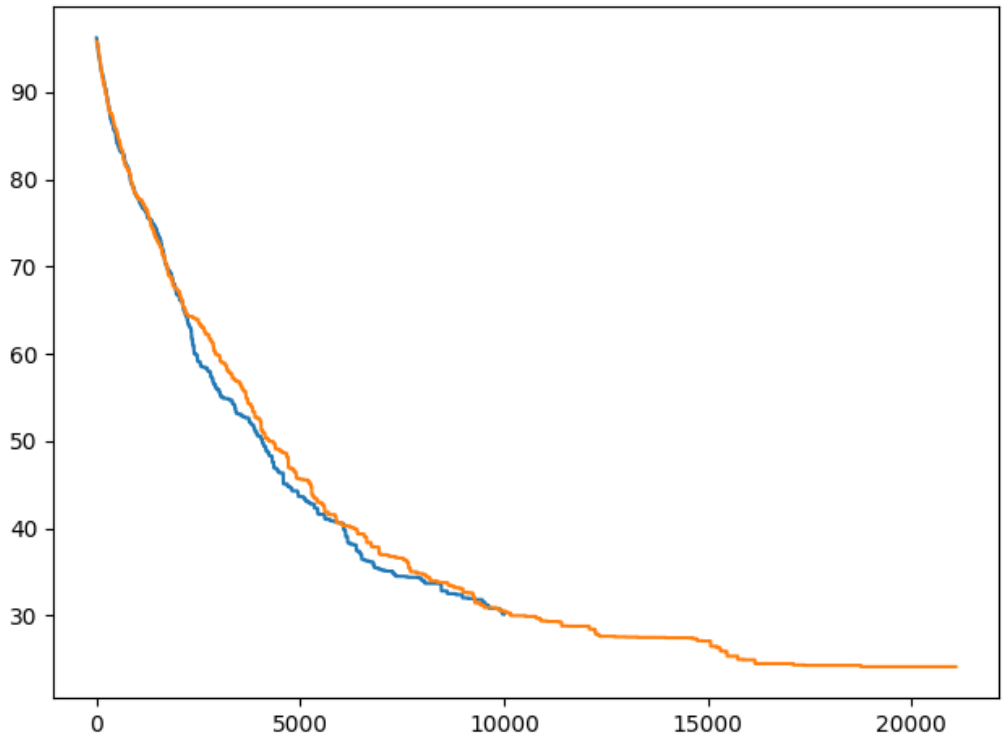
Cuadro 1: BMB (10 % de restricciones)

	Iris				**Ecoli**				**Rand**				**NewThyroid**			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	0.67	0.00	0.67	9.98	22.22	167.00	24.49	91.96	0.76	0.00	0.76	8.42	14.29	0.00	14.29	24.83
Seed 456	0.67	0.00	0.67	9.68	22.55	142.00	24.48	92.23	0.76	0.00	0.76	7.76	14.29	0.00	14.29	26.03
Seed 789	0.67	0.00	0.67	10.26	22.79	161.00	24.99	95.81	0.76	0.00	0.76	7.89	14.29	0.00	14.29	26.71
Seed 101112	0.67	0.00	0.67	10.45	22.54	127.00	24.27	91.68	0.76	0.00	0.76	8.11	14.29	0.00	14.29	26.21
Seed 131415	0.67	0.00	0.67	9.66	22.12	159.00	24.29	91.44	0.76	0.00	0.76	7.89	14.29	0.00	14.29	31.14
Media	0.67	0.00	0.67	10.01	22.44	151.20	24.50	92.62	0.76	0.00	0.76	8.01	14.29	0.00	14.29	26.98

Cuadro 2: BMB (20 % de restricciones)

La mayoría de los *dataset* obtienen buenos resultados, sin embargo Ecoli obtiene resultados peores a los habituales. A parte de posibles diferencias con el espacio de búsqueda, una de las mayores diferencias es el número de iteraciones que se necesitan para llegar a la solución alcanzada por una BL.

Se puede realizar un pequeño experimento para confirmar si las iteraciones es la causa de este problema. Se crea una gráfica con una ejecución con 10.000 evaluaciones y otra con 100.000.



Se puede apreciar que para este conjunto de datos se necesita alrededor de 20.000 evaluaciones con BL para que termine de converger en un óptimo local. Tambien se puede apreciar que por lo general 10.000 iteraciones no llegan cerca de las soluciones obtenidas en las tablas. Por lo tanto, también se puede deducir al gran diferencia que existe al iniciar el algoritmo en diferentes puntos del espacio de búsqueda.

6.2.2. Análisis del Enfriamiento Simulado

En el caso del enfriamiento simulado existe una gran variedad de parámetros que pueden alterar de forma radical el comportamiento del algoritmo. Ya se describió la fórmula utilizada para calcular la temperatura inicial del problema. La temperatura final se establece como $T_f = 0.01$. Además, se han implementado los dos mecanismos de enfriamiento descritos: descenso geométrico y Cauchy modificado. La velocidad de enfriamiento se impone con límites en el máximo número de vecinos generados y aceptados. Por ahora toman los siguientes valores: $MaxGenerated = 10 * n$ y $MaxAccepted = n$. También se impuso una condición extra de parada en la que el número de vecinos aceptados en cada iteración debe ser mayor a cero.

Se toman medidas con 3 diferentes mecanismos de enfriamiento. El enfriamiento de Cauchy permite establecer aproximadamente un número de iteraciones independiente de las temperaturas iniciales o finales. Sin embargo, este mecanismo a veces provoca demasiada convergencia al enfriar muy rápido. Por ello, se ha decidido utilizar también el descenso geométrico con una constante alta para favorecer la exploración. Se han tomado dos valores $\alpha = 0.95$ y $\alpha = 0.98$ ya que provocan un número de evaluaciones cercano al límite.

	Iris				**Ecoli**				**Rand**				**NewThyroid**			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	0.67	0.00	0.67	41.69	21.69	87.00	24.10	85.99	0.76	0.00	0.76	27.27	13.83	6.00	14.07	42.59
Seed 456	0.67	0.00	0.67	40.57	21.69	72.00	23.68	85.31	0.76	0.00	0.76	29.53	13.83	6.00	14.07	27.97
Seed 789	0.67	0.00	0.67	40.79	21.71	73.00	23.73	85.43	0.76	0.00	0.76	24.57	13.83	6.00	14.07	29.98
Seed 101112	0.67	0.00	0.67	41.32	21.63	86.00	24.01	56.53	0.76	0.00	0.76	23.80	13.83	6.00	14.07	30.11
Seed 131415	0.67	0.00	0.67	40.79	22.02	67.00	23.87	84.49	0.76	0.00	0.76	28.36	10.87	98.00	14.63	19.09
Media	0.67	0.00	0.67	41.03	21.75	77.00	23.88	79.55	0.76	0.00	0.76	26.71	13.24	24.40	14.18	29.95

Cuadro 3: ES Cauchy (10 % de restricciones)

	Iris				**Ecoli**				**Rand**				**NewThyroid**			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	0.67	0.00	0.67	41.39	21.87	157.00	24.01	83.92	0.76	0.00	0.76	27.38	14.29	0.00	14.29	93.75
Seed 456	0.67	0.00	0.67	41.73	21.94	124.00	23.63	81.48	0.76	0.00	0.76	26.31	10.89	233.00	15.26	8.16
Seed 789	0.67	0.00	0.67	40.18	21.79	177.00	24.20	83.85	0.76	0.00	0.76	27.48	14.29	0.00	14.29	95.54
Seed 101112	0.67	0.00	0.67	35.09	21.95	148.00	23.97	83.15	0.76	0.00	0.76	25.86	10.89	234.00	15.27	8.02
Seed 131415	0.67	0.00	0.67	40.51	21.81	147.00	23.81	83.24	0.76	0.00	0.76	20.90	14.29	0.00	14.29	93.32
Media	0.67	0.00	0.67	39.78	21.87	150.60	23.92	83.13	0.76	0.00	0.76	25.59	12.93	93.40	14.68	59.76

Cuadro 4: ES Cauchy (20 % de restricciones)

	Iris				**Ecoli**				**Rand**				**NewThyroid**			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	0.67	0.00	0.67	25.14	21.74	78.00	23.90	91.55	0.76	0.00	0.76	23.37	13.83	6.00	14.07	90.68
Seed 456	0.67	0.00	0.67	25.54	21.81	71.00	23.77	93.99	0.76	0.00	0.76	19.74	13.83	6.00	14.07	97.28
Seed 789	0.67	0.00	0.67	24.58	21.44	98.00	24.15	91.21	0.76	0.00	0.76	21.46	13.83	6.00	14.07	96.68
Seed 101112	0.67	0.00	0.67	26.10	21.65	91.00	24.17	92.28	0.76	0.00	0.76	23.40	13.83	6.00	14.07	98.66
Seed 131415	0.67	0.00	0.67	26.48	21.74	76.00	23.84	92.88	0.76	0.00	0.76	22.73	13.83	6.00	14.07	90.95
Media	0.67	0.00	0.67	25.57	21.68	82.80	23.97	92.38	0.76	0.00	0.76	22.14	13.83	6.00	14.07	94.85

Cuadro 5: ES Geo 0.95 (10 % de restricciones)

	Iris				**Ecoli**				**Rand**				**NewThyroid**			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	0.67	0.00	0.67	26.23	21.87	141.00	23.79	96.88	0.76	0.00	0.76	21.87	14.29	0.00	14.29	102.32
Seed 456	0.67	0.00	0.67	25.37	22.00	157.00	24.14	94.86	0.76	0.00	0.76	24.05	14.29	0.00	14.29	102.70
Seed 789	0.67	0.00	0.67	25.84	21.88	157.00	24.02	98.16	0.76	0.00	0.76	20.05	14.29	0.00	14.29	102.46
Seed 101112	0.67	0.00	0.67	25.21	21.73	169.00	24.03	97.28	0.76	0.00	0.76	21.73	14.29	0.00	14.29	101.16
Seed 131415	0.67	0.00	0.67	24.63	19.57	268.00	23.22	94.55	0.76	0.00	0.76	20.04	14.29	0.00	14.29	100.58
Media	0.67	0.00	0.67	25.45	21.41	178.40	23.84	96.35	0.76	0.00	0.76	21.55	14.29	0.00	14.29	101.85

Cuadro 6: ES Geo 0.95 (20 % de restricciones)

	Iris				Ecoli				Rand				NewThyroid			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	0.67	0.00	0.67	59.09	41.64	1499.00	83.10	75.67	0.76	0.00	0.76	50.16	13.83	6.00	14.07	93.57
Seed 456	0.67	0.00	0.67	60.81	42.77	1450.00	82.87	76.31	0.76	0.00	0.76	48.77	13.55	15.00	14.12	95.43
Seed 789	0.67	0.00	0.67	59.48	41.99	1408.00	80.93	75.53	0.76	0.00	0.76	49.39	13.83	6.00	14.07	93.83
Seed 101112	0.67	0.00	0.67	58.53	40.21	1455.00	80.45	78.03	0.76	0.00	0.76	48.89	13.83	6.00	14.07	94.20
Seed 131415	0.67	0.00	0.67	59.41	41.41	1422.00	80.73	76.76	0.76	0.00	0.76	48.47	13.83	6.00	14.07	93.06
Media	0.67	0.00	0.67	59.46	41.60	1446.80	81.62	76.46	0.76	0.00	0.76	49.14	13.78	7.80	14.08	94.02

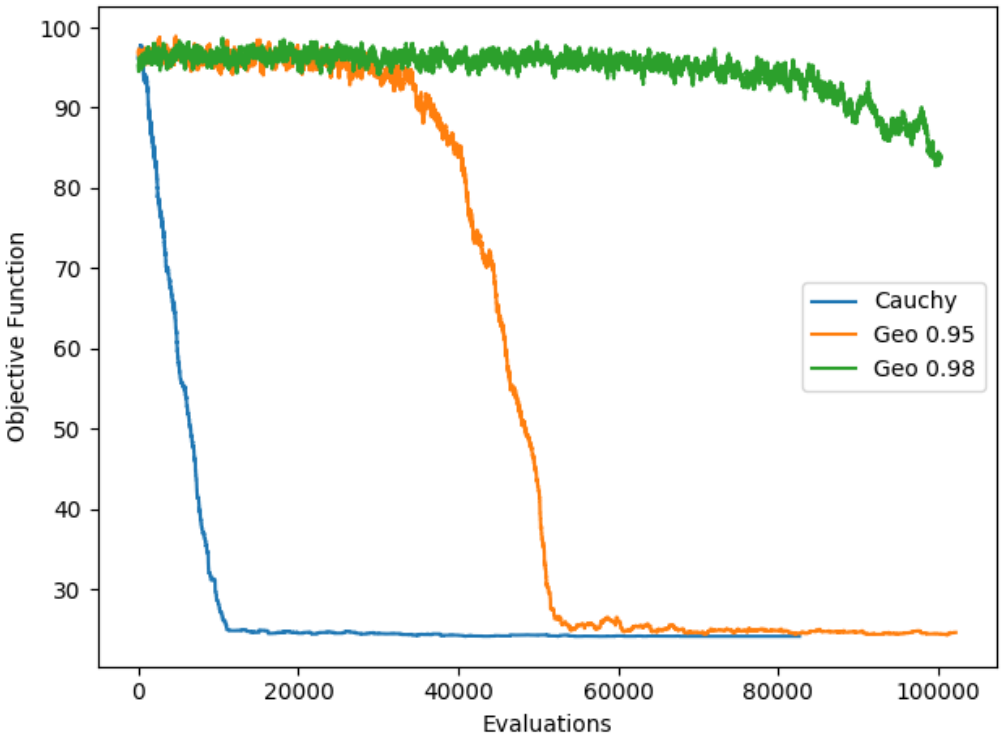
Cuadro 7: ES Geo 0.98 (10 % de restricciones)

	Iris				Ecoli				Rand				NewThyroid			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	0.67	0.00	0.67	61.05	41.17	2019.00	68.67	79.30	0.76	0.00	0.76	49.67	14.29	0.00	14.29	97.90
Seed 456	0.67	0.00	0.67	61.57	38.73	2479.00	72.49	77.77	0.76	0.00	0.76	48.44	14.29	0.00	14.29	96.25
Seed 789	0.67	0.00	0.67	60.94	40.25	2719.00	77.28	81.07	0.76	0.00	0.76	47.69	14.29	0.00	14.29	96.13
Seed 101112	0.67	0.00	0.67	61.51	39.66	2849.00	78.46	78.28	0.76	0.00	0.76	49.62	14.29	0.00	14.29	96.70
Seed 131415	0.67	0.00	0.67	59.35	37.50	2510.00	71.68	79.76	0.76	0.00	0.76	49.28	14.29	0.00	14.29	96.15
Media	0.67	0.00	0.67	60.88	39.46	2515.20	73.72	79.24	0.76	0.00	0.76	48.94	14.29	0.00	14.29	96.62

Cuadro 8: ES Geo 0.98 (20 % de restricciones)

Se obtienen resultados similares para el esquema de Cauchy y el primer descenso geométrico. Sin embargo, el segundo descenso geométrico obtiene valores mucho peores para el conjunto Ecoli. Esto puede ser por el límite de evaluaciones impuesto. Se sabe que al aumentar la constante α se provoca una mayor exploración del entorno. Si existe demasiada exploración, se gastan todas las evaluaciones antes de llegar a una solución válida.

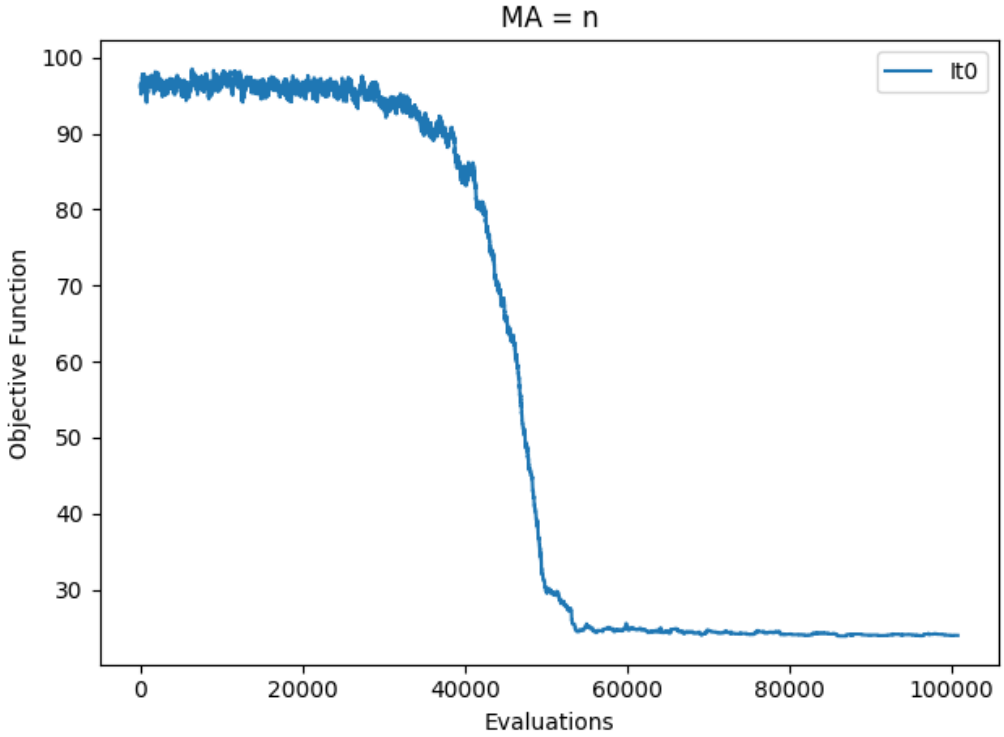
Para confirmar esta hipótesis se traslada la convergencia a una gráfica:

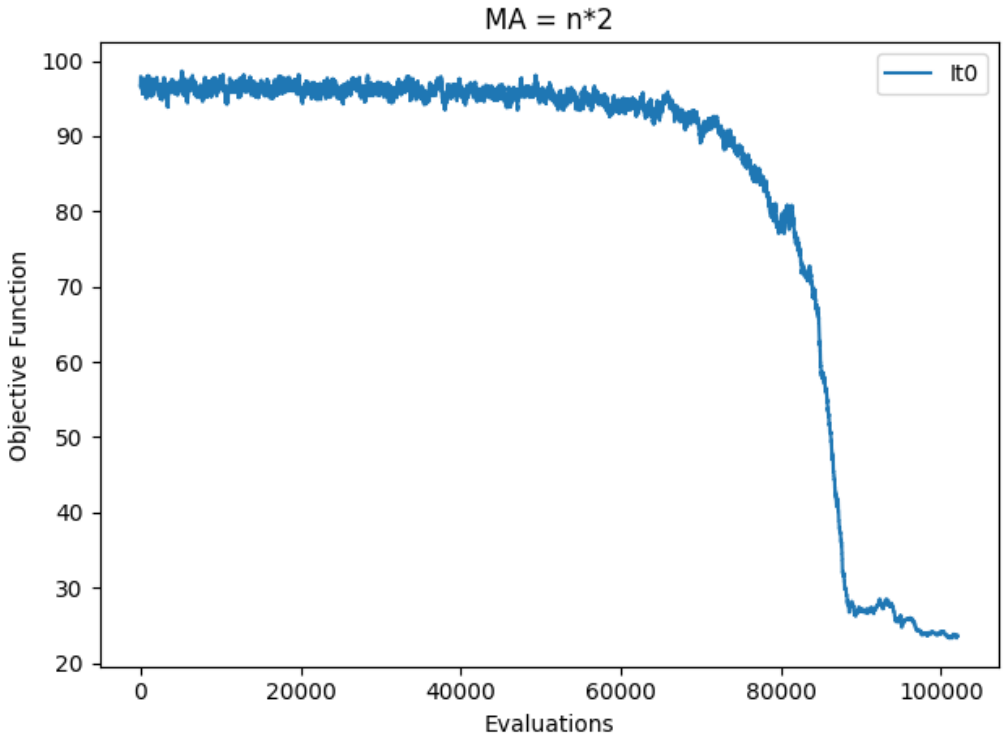


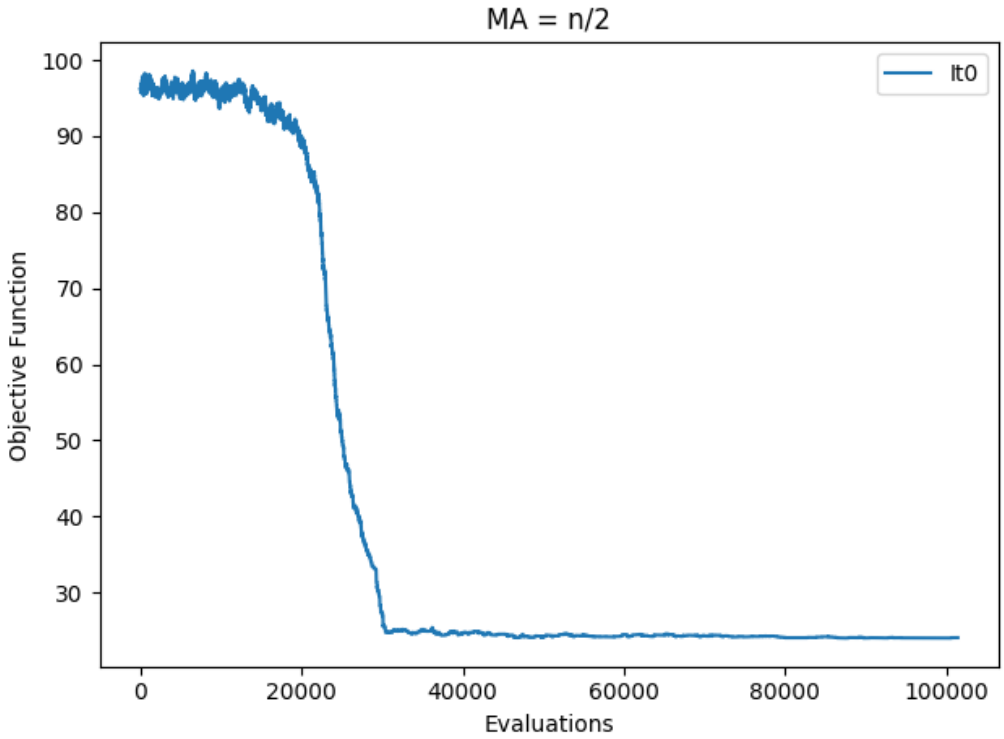
Al observar la gráfica se puede confirmar la hipótesis propuesta. El segundo mecanismo de enfriamiento provoca demasiada exploración del entorno, lo que provoca la admisión de demasiadas soluciones peores a la actual. La convergencia no es lo suficientemente rápida como para hallar una solución buena con el número de evaluaciones dado. También se puede observar con el enfriamiento con el esquema de Cauchy es mucho más rápido que con el descenso geométrico. Al enfriar de forma tan brusca al comienzo provoca que no haya una exploración del entorno y la temperatura baje rápidamente. Cuando la temperatura es lo suficientemente baja, el algoritmo no aceptará tantas soluciones peores a la actual. Esto significa que se comienzan a aceptar una mayor proporción de soluciones mejores a la actual lo que provoca la convergencia tan rápida.

6.2.2.1. Análisis de parámetros

Se ha hablado anteriormente sobre la influencia de cada uno de los parámetros del algoritmo en su funcionamiento. Se realiza entonces un experimento para analizar los efectos de diferentes valores en la velocidad de enfriamiento. Se modifican entonces el número máximo de vecinos generados y aceptados. Se comienza primero con el máximo de vecinos aceptados:

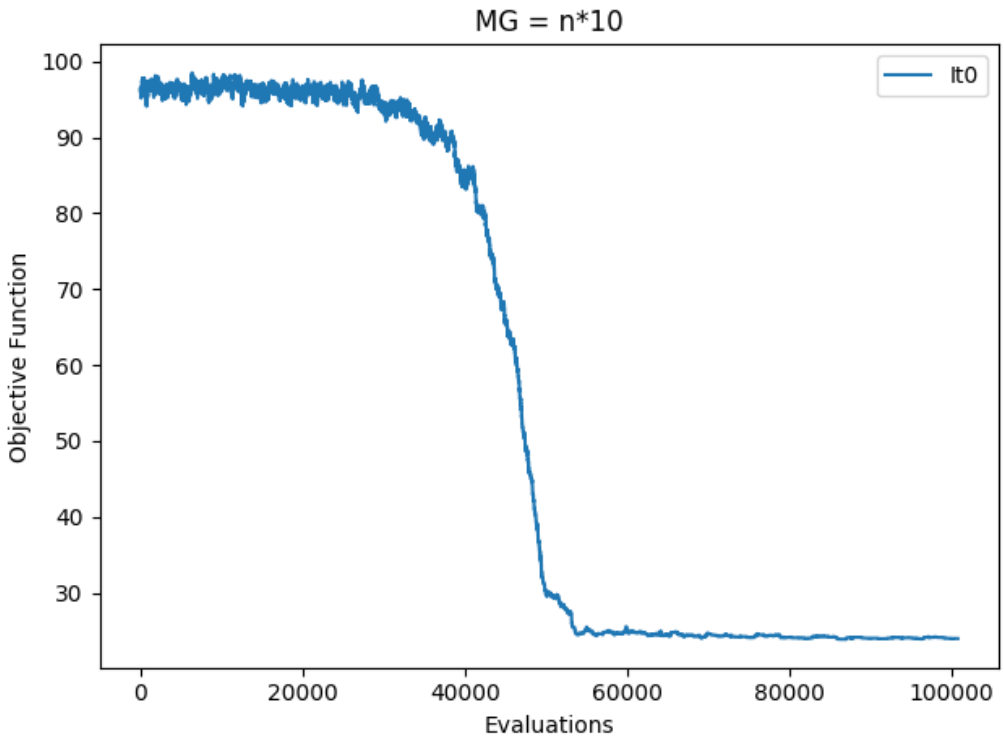

MA = n
It0
Objective Function
Evaluations
100
90
80
70
60
50
40
30
0
20000
40000
60000
80000
100000

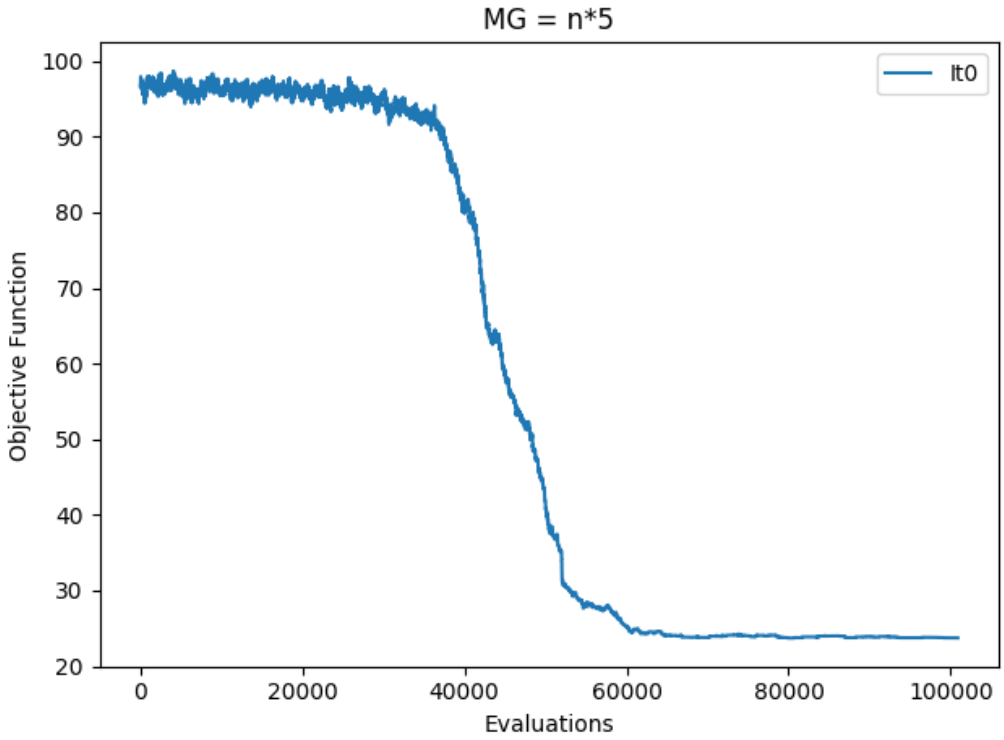

MA = n*2
It0
Objective Function
Evaluations
100
90
80
70
60
50
40
30
20
0
20000
40000
60000
80000
100000

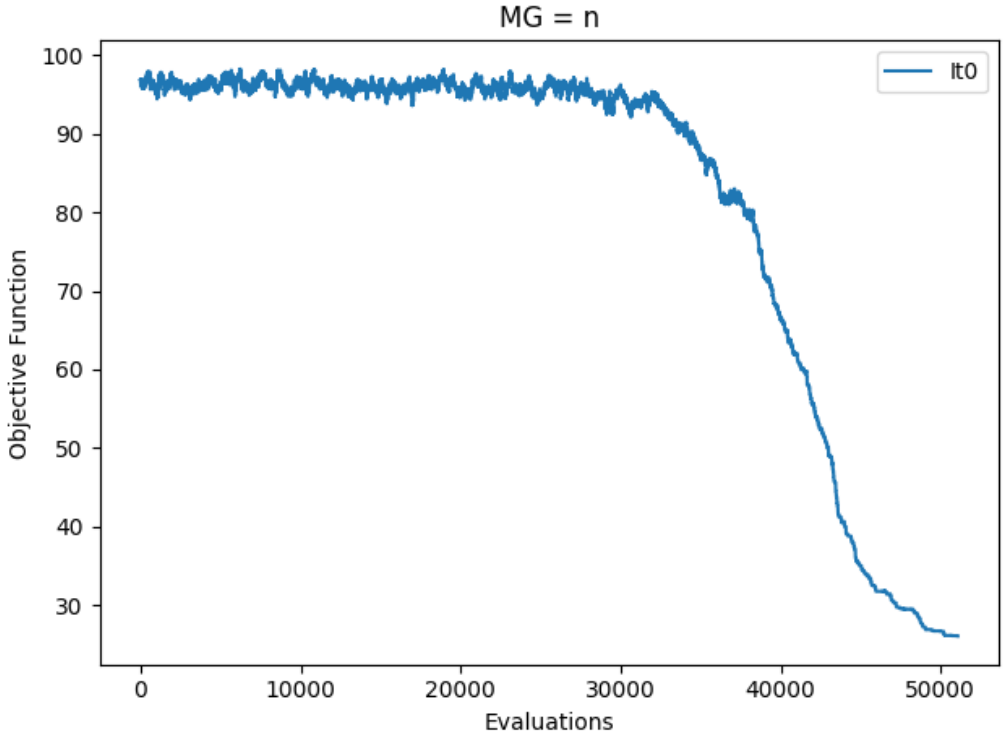

MA = n/2
It0
Objective Function
Evaluations
100
90
80
70
60
50
40
30
0
20000
40000
60000
80000
100000

Se comienza con una gráfica con el valor recomendado y utilizado en el anterior estudio. Se puede ver que existe una gran diferencia entre las gráficas. Un mayor número de vecinos aceptados provoca que comience a converger mucho mas tarde. Al comienzo del programa la temperatura es alta, por lo que esta provoca que se acepten soluciones peores que la actual. El aumento de este límite provoca una fuerte exploración al comienzo. Todas las variaciones convergen de forma rápida cuando comienzan a hacerlo. Esto se puede deber a que existe un área en el espacio de búsqueda con mejores soluciones. Se puede apreciar cierto repunte al final de la convergencia en el primer mecanismo. Esto puede ser provocado por haber alcanzado un óptimo local. Mientras que la velocidad de enfriamiento por defecto realiza una ligera exploración antes de terminar de converger, al aceptar mayor número de vecinos esta última exploración también se ve aumentada. Por otra parte, al aceptar menos vecinos el problema se queda estancado en este óptimo local.

Se pasa entonces a analizar el comportamiento del número máximo de vecinos generados. Se aclara que solo se ha modificado el número máximo de vecinos generados y se ha mantenido el número máximo de vecinos aceptados:

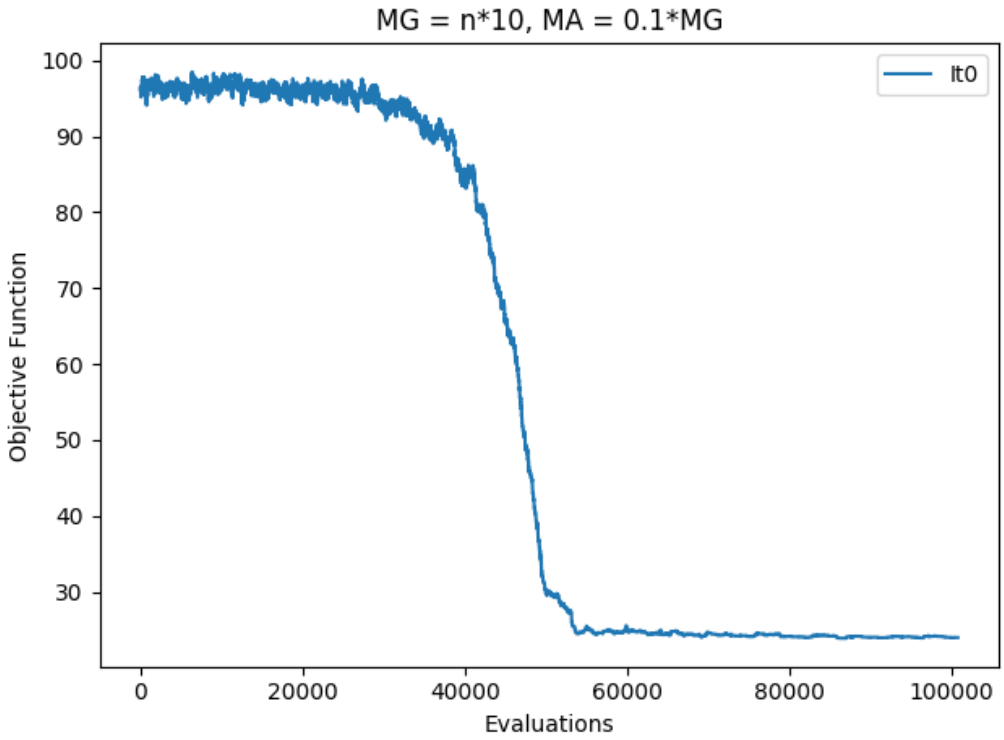


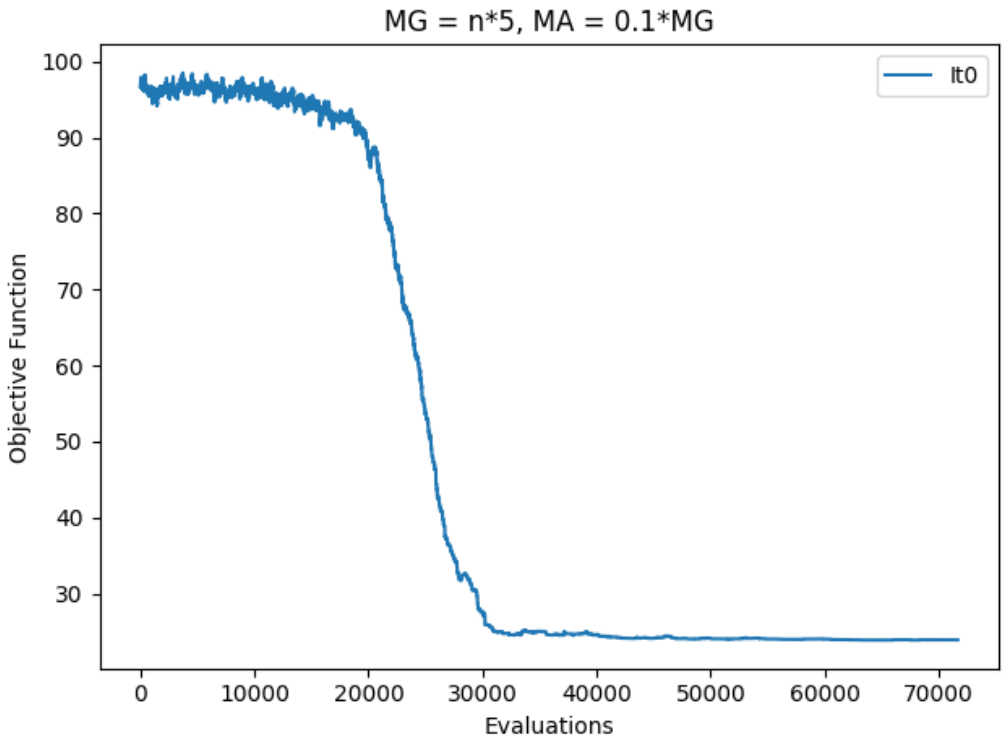

MG = n*5
It0
Objective Function
Evaluations
100
90
80
70
60
50
40
30
20
0
20000
40000
60000
80000
100000

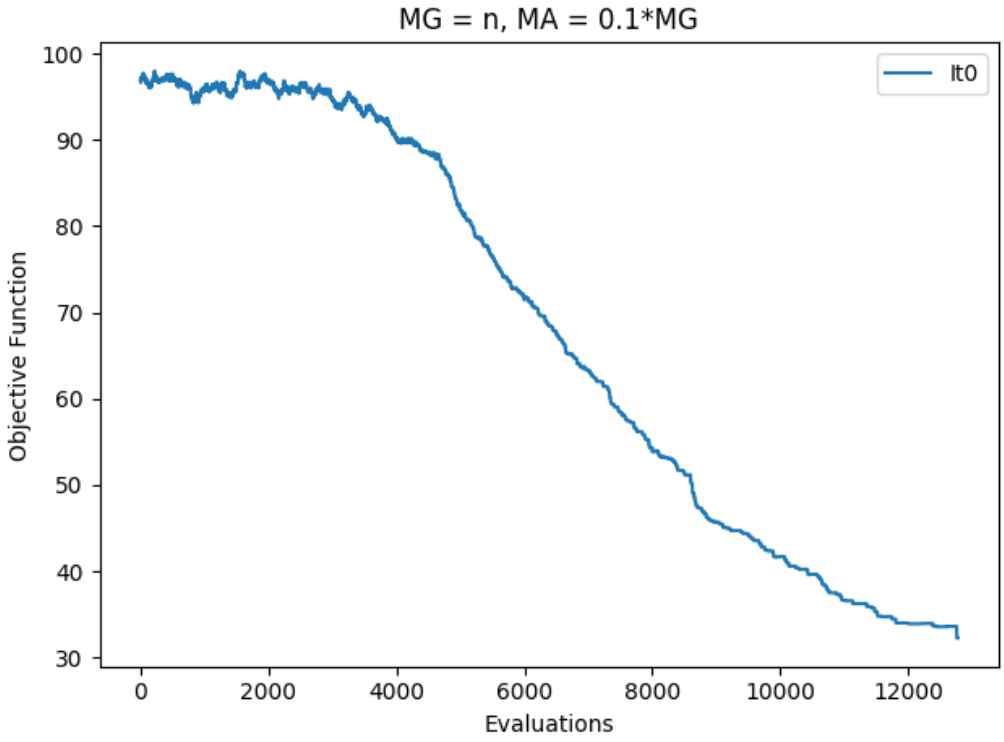

MG = n
It0
Objective Function
Evaluations
100
90
80
70
60
50
40
30
0
10000
20000
30000
40000
50000

No se aprecian cambios notables entre las dos primeras gráficas. La convergencia es simplemente algo menor y existen ciertos repuntes en esta. Sin embargo, la última gráfica sufre muchos cambios. Al reducir tanto el máximo de vecinos generados y mantener el máximo de aceptados se llega a que $MA = MG$. Esto provoca demasiada convergencia lo que hace terminar el programa prematuramente.

Se ha observado que en la última comparación no existian demasiados cambios o estos eran muy bruscos. Esto puede ser al perder la proporcionalidad entre MA y MG. Por ello se realiza un último experimento donde se volverán a modificar los valores de MG pero manteniendo la proporcionalidad $MA = MG * 0.1$.




MG = n*5, MA = 0.1*MG
It0
Objective Function
Evaluations
100
90
80
70
60
50
40
30
0
10000
20000
30000
40000
50000
60000
70000


MG = n, MA = 0.1*MG
It0
Objective Function
Evaluations
100
90
80
70
60
50
40
30
0
2000
4000
6000
8000
10000
12000

Aunque las primeras gráficas sean parecidas existen grandes diferencias. La segunda gráfica comienza a converger en el mismo punto que en el experimento $MA = n/2$. Si se desarrolla, se podrá ver que de hecho $MA = 0.5 * n = n/2$. Pero existe otra diferencia, la función termina de converger mucho mas rápido que anteriormente. No ha utilizado casi 30.000 evaluaciones y alcanza el mismo resultado. Con respecto a la tercera gráfica, se aprecia una convergencia todavía más rápida. El efecto de reducir el máximo de vecinos generados provocaba una gran convergencia. El efecto de reducir el máximo de vecinos aceptados reducía la exploración al comienzo del programa. Al mantener la proporcionalidad $MA = n/10$ aumenta aún más los efectos anteriores.Cuando se combinan ambos se consigue un programa que explora muy brevemente al comienzo y converge muy rápidamente a la solución final. Sin embargo, se ha de notar que ha alcanzado una solución cercana a los 30 puntos cuando el resto de variaciones seguían alrededor de 95 puntos con el mismo número de evaluaciones.

6.2.3. Análisis de la Búsqueda Iterativa

La búsqueda iterativa contiene pocos parámetros modificables pero cada uno toma un papel muy importante en el comportamiento de este algoritmo. Se van a discutir dos posibles modificaciones con resultados y comportamientos muy diferentes. Para ambos casos la solución inicial será generada aleatoriamente. También se ha descrito el proceso de modificación elegido anteriormente. El criterio de aceptación también sera igual para ambos, eligiendo la mejor solución. Sin embargo, se diferencian en el procedimiento de búsqueda usado. Uno de los casos empleará la búsqueda local mientras que el otro usa el enfriamiento simulado.

6.2.3.1. Búsqueda Iterativa Básica

En este caso se utiliza la búsqueda local como procedimiento de búsqueda. Esto puede permitir superar alguna de las limitaciones de la BMB. Mantener soluciones parciales anteriores permite mantener arrancar desde distintos puntos del espacio de soluciones y, a la vez, mantener cierta parte de la solución calculada anteriormente. La exploración será mucho menor que en la BMB pero puede ayudar a mejorar los resultados si el espacio de búsqueda tiene varios óptimos locales cerca del óptimo global. De esta forma obtenemos los siguientes resultados:

	Iris				**Ecoli**				**Rand**				**NewThyroid**			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	0.67	0.00	0.67	6.16	21.49	87.00	23.89	97.57	0.76	0.00	0.76	4.70	10.81	98.00	14.56	21.16
Seed 456	0.67	0.00	0.67	6.27	21.29	86.00	23.67	96.04	0.76	0.00	0.76	5.04	10.88	97.00	14.60	17.38
Seed 789	0.67	0.00	0.67	5.67	21.54	80.00	23.75	98.52	0.76	0.00	0.76	5.17	13.83	6.00	14.07	15.29
Seed 101112	0.67	0.00	0.67	6.63	21.28	86.00	23.66	99.62	0.76	0.00	0.76	5.15	13.83	6.00	14.07	15.86
Seed 131415	0.67	0.00	0.67	6.30	22.06	54.00	23.56	94.68	0.76	0.00	0.76	5.23	10.87	96.00	14.56	20.01
Media	0.67	0.00	0.67	6.20	21.53	78.60	23.71	97.28	0.76	0.00	0.76	5.06	12.05	60.60	14.37	17.94

Cuadro 9: ILS (10 % de restricciones)

	Iris				Ecoli				Rand				NewThyroid			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	0.67	0.00	0.67	6.05	21.97	115.00	23.53	94.85	0.76	0.00	0.76	5.13	10.88	229.00	15.17	22.18
Seed 456	0.67	0.00	0.67	6.71	21.99	119.00	23.61	98.22	0.76	0.00	0.76	5.06	10.78	233.00	15.14	18.74
Seed 789	0.67	0.00	0.67	6.00	21.76	129.00	23.52	102.20	0.76	0.00	0.76	4.98	14.29	0.00	14.29	15.30
Seed 101112	0.67	0.00	0.67	5.60	22.08	113.00	23.62	106.25	0.76	0.00	0.76	4.77	14.29	0.00	14.29	17.21
Seed 131415	0.67	0.00	0.67	6.47	22.04	123.00	23.72	104.83	0.76	0.00	0.76	5.00	10.88	229.00	15.17	17.81
Media	0.67	0.00	0.67	6.17	21.97	119.80	23.60	101.27	0.76	0.00	0.76	4.99	12.22	138.20	14.81	18.25

Cuadro 10: ILS (20 % de restricciones)

De nuevo se alcanzan valores óptimos en los conjuntos de datos pequeños. Aunque en Ecoli se alcanzan valores algo más pequeños de lo normal con respecto a *infeasibility*. Ya se vio anteriormente que el algoritmo BL alcanzaba valores de *infeasibility* bajos. Se puede suponer que este algoritmo también alcanza estos valores pero gracias a la fuerte mutación, le permite salir de los óptimos locales. Esto provoca un aumento en *infeasibility* pero una mejora en los valores de la función objetivo.

6.2.3.2. Búsqueda Iterativa Híbrida

En este caso se utiliza la el algoritmo del enfriamiento simulado como proceso de búsqueda. Esto combinación añade principalmente convergencia al proceso de búsqueda. También añade cierta exploración gracias al proceso de mutación. Esto produce una buena combinación que puede permitir llegar a nuevas soluciones. Sin embargo, el número de parámetros de ambos algoritmos también se combina. Esto provoca que pueda ser complejo elegir los valores adecuados para cada conjunto. Por esto se vuelven a utilizar los tres diferentes mecanismos de enfriamiento. Se exponen los resultados:

	Iris				**Ecoli**				**Rand**				**NewThyroid**			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	0.67	0	0.67	31.57	21.89	77	24.01	37.49	0.76	0	0.76	29.98	13.83	6	14.07	57.01
Seed 456	0.67	0	0.67	30.6	23.92	78	26.08	36.95	0.76	0	0.76	27.08	13.83	6	14.07	57.22
Seed 789	0.67	0	0.67	31.61	24.13	81	26.37	38.39	0.76	0	0.76	28.1	13.83	6	14.07	55.34
Seed 101112	0.67	0	0.67	31.51	22.15	89	24.61	39.82	0.76	0	0.76	27.71	13.83	6	14.07	58.18
Seed 131415	0.67	0	0.67	30.13	24.64	116	27.85	36.58	0.76	0	0.76	28.1	13.83	6	14.07	54.99
Media	0.67	0	0.67	31.08	23.35	88.2	25.79	37.85	0.76	0	0.76	28.19	13.83	6	14.07	56.55

Cuadro 11: ILS-ES Cauchy (10 % de restricciones)

	Iris				**Ecoli**				**Rand**				**NewThyroid**			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	0.67	0	0.67	30.56	24.1	157	26.24	38.49	0.76	0	0.76	28.82	14.29	0	14.29	85.64
Seed 456	0.67	0	0.67	30.63	22.41	190	25	40.06	0.76	0	0.76	28.25	14.29	0	14.29	75.63
Seed 789	0.67	0	0.67	30.51	22.86	219	25.84	40.37	0.76	0	0.76	26.44	14.29	0	14.29	82.38
Seed 101112	0.67	0	0.67	31.28	22.19	186	24.72	40.41	0.76	0	0.76	30.08	14.29	0	14.29	72.36
Seed 131415	0.67	0	0.67	31.06	22	162	24.2	39.71	0.76	0	0.76	30.74	14.29	0	14.29	83.36
Media	0.67	0	0.67	30.81	22.71	182.8	25.2	39.81	0.76	0	0.76	28.87	14.29	0	14.29	79.87

Cuadro 12: ILS-ES Cauchy (20 % de restricciones)

	Iris				**Ecoli**				**Rand**				**NewThyroid**			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	1.54	340.00	3.85	60.28	45.31	1814.00	95.48	75.82	2.71	447.00	6.16	49.79	12.91	1138.00	56.56	88.51
Seed 456	1.84	414.00	4.66	57.82	45.18	1808.00	95.18	75.60	2.65	420.00	5.89	46.98	13.09	1123.00	56.17	87.69
Seed 789	1.74	425.00	4.63	57.75	45.95	1796.00	95.62	75.65	2.53	431.00	5.86	47.29	13.21	1119.00	56.14	88.42
Seed 101112	1.64	404.00	4.39	59.17	45.84	1793.00	95.42	76.92	2.64	431.00	5.97	46.92	13.08	1112.00	55.74	88.59
Seed 131415	1.87	401.00	4.60	59.46	45.84	1770.00	94.79	75.14	2.54	412.00	5.72	46.55	13.22	1104.00	55.58	90.38
Media	1.73	396.80	4.43	58.89	45.62	1796.20	95.30	75.83	2.61	428.20	5.92	47.51	13.10	1119.20	56.04	88.72

Cuadro 13: ILS-ES Geo 0.95 (10 % de restricciones)

	Iris				**Ecoli**				**Rand**				**NewThyroid**			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	1.73	850.00	4.51	58.41	45.80	3566.00	94.36	77.97	2.51	851.00	5.68	48.26	12.86	2302.00	55.95	91.39
Seed 456	1.58	774.00	4.12	58.60	45.82	3591.00	94.72	77.99	2.46	830.00	5.55	48.04	12.95	2282.00	55.67	92.27
Seed 789	1.66	876.00	4.54	59.66	45.55	3599.00	94.56	76.95	2.55	837.00	5.67	47.99	13.30	2279.00	55.96	90.61
Seed 101112	1.81	901.00	4.76	60.81	45.81	3576.00	94.51	78.41	2.52	843.00	5.66	51.16	13.04	2275.00	55.63	91.51
Seed 131415	1.82	886.00	4.73	58.77	45.72	3601.00	94.76	78.88	2.68	899.00	6.03	49.20	13.14	2284.00	55.89	92.21
Media	1.72	857.40	4.53	59.25	45.74	3586.60	94.58	78.04	2.54	852.00	5.72	48.93	13.06	2284.40	55.82	91.60

Cuadro 14: ILS-ES Geo 0.95 (20 % de restricciones)

	Iris				Ecoli				Rand				NewThyroid			
	TasaC	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T
Seed 123	1.90	466.00	5.07	58.90	45.74	1782.00	95.02	75.23	2.74	458.00	6.28	47.54	13.06	1136.00	56.64	90.05
Seed 456	1.93	478.00	5.18	56.99	45.67	1771.00	94.65	74.57	2.74	479.00	6.44	48.60	13.06	1151.00	57.21	90.24
Seed 789	1.86	443.00	4.87	57.43	45.92	1791.00	95.45	75.51	2.77	478.00	6.47	47.73	13.04	1137.00	56.66	90.24
Seed 101112	1.86	468.00	5.04	58.80	45.19	1824.00	95.63	76.18	2.76	470.00	6.39	46.15	13.13	1136.00	56.71	89.94
Seed 131415	1.94	475.00	5.17	58.31	45.85	1775.00	94.93	74.44	2.73	485.00	6.48	46.57	13.00	1136.00	56.58	89.42
Media	1.90	466.00	5.07	58.09	45.67	1788.60	95.14	75.18	2.75	474.00	6.41	47.32	13.06	1139.20	56.76	89.98

Cuadro 15: ILS-ES Geo 0.98 (10 % de restricciones)

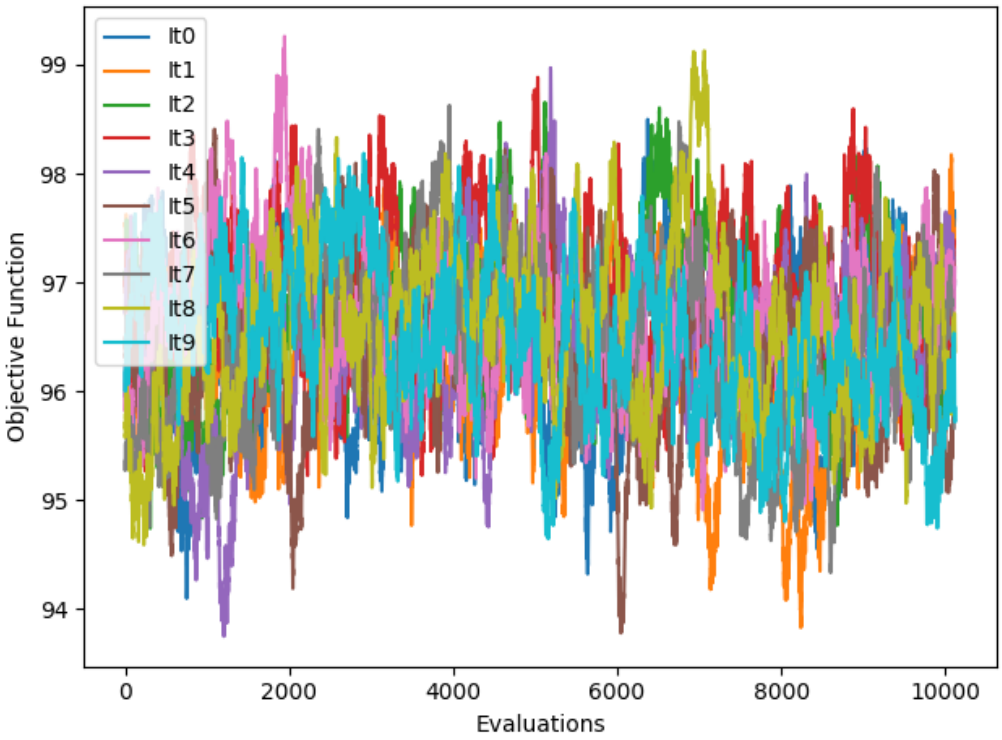
	Iris				Ecoli				Rand				NewThyroid			
	TasaC	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T
Seed 123	1.90	946.00	5.01	58.60	45.83	3605.00	94.92	76.94	2.73	949.00	6.27	48.82	12.70	2309.00	55.93	89.11
Seed 456	1.94	953.00	5.06	61.28	45.48	3651.00	95.20	76.77	2.79	956.00	6.35	47.92	13.05	2311.00	56.31	92.11
Seed 789	1.94	970.00	5.12	59.86	45.48	3568.00	94.07	78.32	2.71	929.00	6.17	47.72	13.11	2298.00	56.13	91.53
Seed 101112	1.89	955.00	5.02	59.22	45.45	3595.00	94.41	76.30	2.64	935.00	6.13	48.15	13.14	2264.00	55.52	91.34
Seed 131415	1.91	952.00	5.03	59.32	45.44	3582.00	94.22	77.36	2.79	951.00	6.33	48.19	12.98	2292.00	55.89	90.93
Media	1.92	955.20	5.05	59.66	45.54	3600.20	94.56	77.14	2.73	944.00	6.25	48.16	13.00	2294.80	55.96	91.00

Cuadro 16: ILS-ES Geo 0.98 (20 % de restricciones)

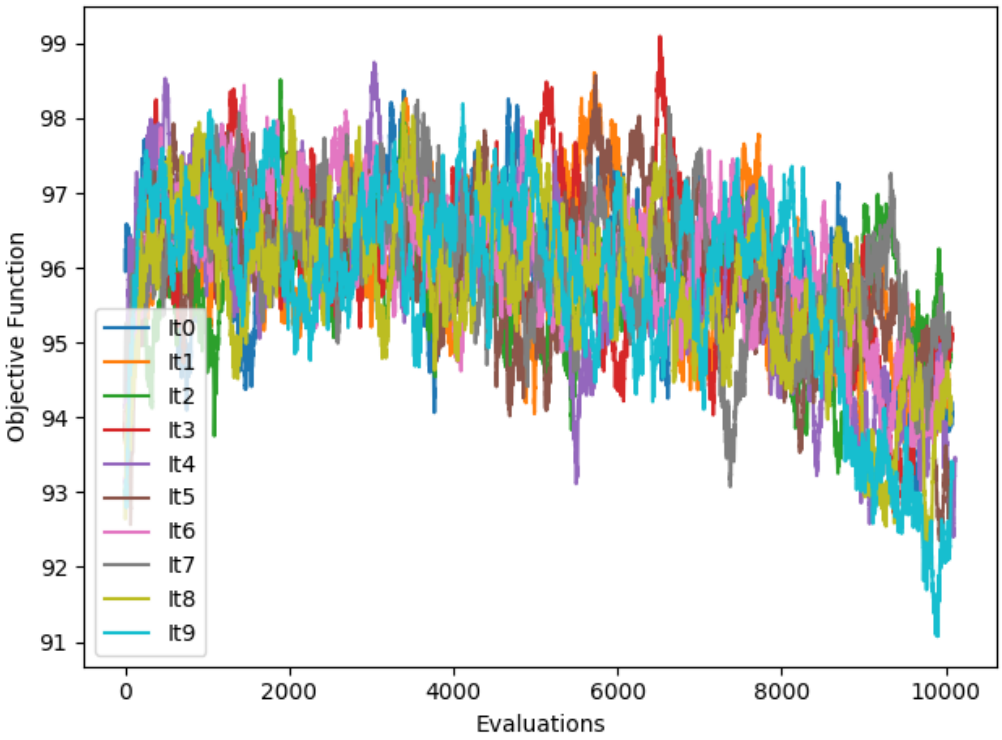
Se puede ver que solo el esquema Cauchy trae resultados aceptables. Se ha visto anteriormente como los otros esquemas pueden tardar más en converger. La diferencia con resultados anteriores es que tampoco converge en los conjuntos de datos pequeños.

6.2.3.3. Análisis de parámetros

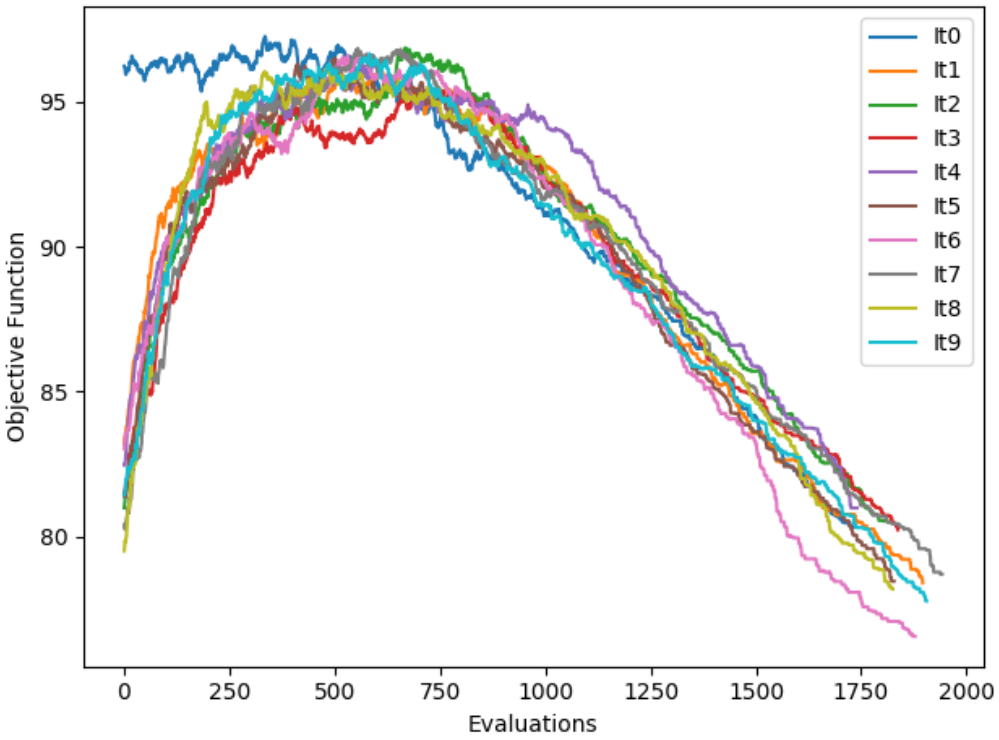
Como una forma de aplicar conocimientos obtenidos en anteriores experimentos, se intentaran modificar los parámetros necesarios en ILS-ES para obtener resultados aceptables con un esquema de descenso geométrico. Para esto se irán modificando los elementos y valores uno a uno mientras se exponen en una gráfica los resultados. Se utiliza el conjunto de datos Ecoli por ser el más descriptivo. Se comienza entonces con el caso estudiado con $\alpha = 0.95$.



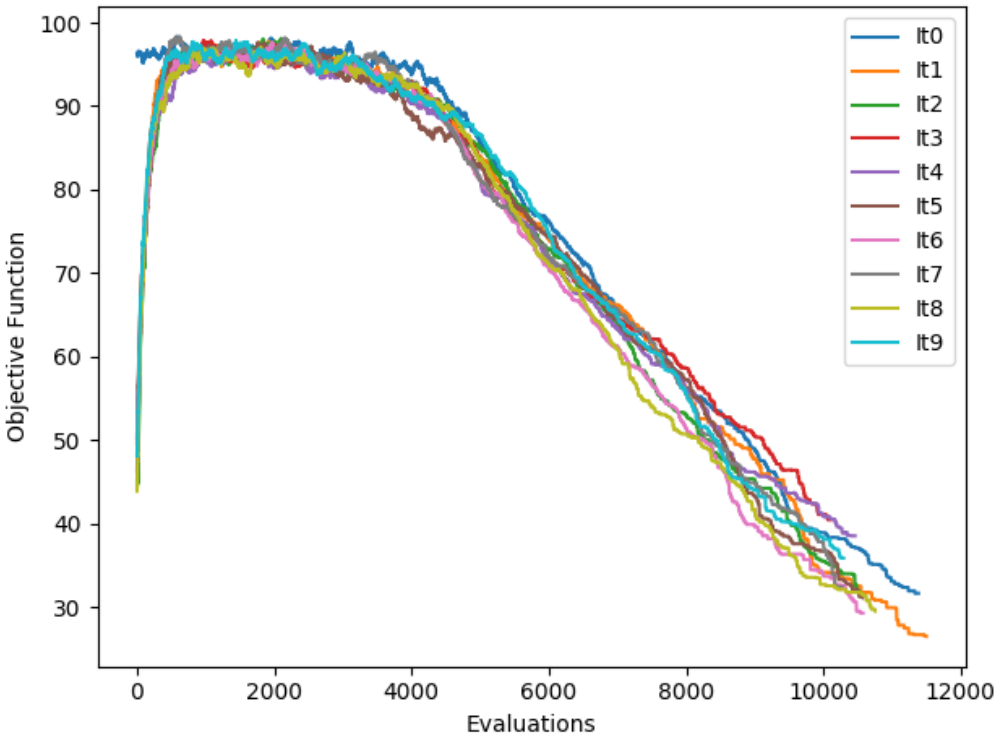
Como se puede ver, no se va a obtener ningún buen resultado ya que el algoritmo no converge. Comienza y termina con valores similares. Se puede remover exploración reduciendo la constante del esquema de enfriamiento. Se cambia entonces a $\alpha = 0.8$ obteniendo:



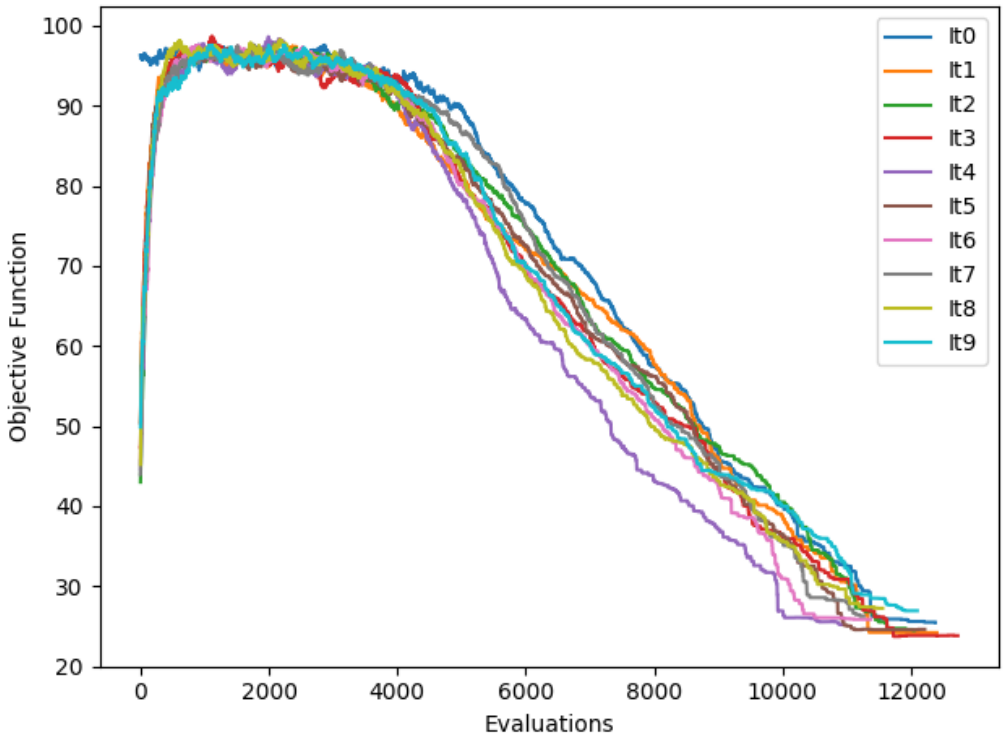
Se puede ver como existe algo más de convergencia en la función. Sin embargo, sigue habiendo demasiada variación en los valores. Se sigue necesitando imponer mucha más convergencia e intentar reducir la temperatura para así reducir la exploración. Ya que la constante está cercana del límite inferior recomendado se pasa a otros parámetros. En el anterior análisis de parámetros se discutía como al reducir el número máximo de número de vecinos generados y el de vecinos aceptados (proporcionalmente) provocaba convergencia. Dado que se necesita mucha convergencia aún se utilizan los valores con mayor convergencia, es decir $MG = n, MA = 0.1 * MG$ con lo que se obtiene:



El problema ahora es que converge demasiado rápido y activa la condición de parada sin usar la mayoría de las evaluaciones. Se cambian los parámetros a otra de las variaciones usadas anteriormente $MG = 5 * n, MA = 0.1 * MG$ para aumentar la exploración y se obtiene:



Se consiguen entonces resultados aceptables para el conjunto de datos. Sabiendo el ratio aceptable de convergencia, se pueden alterar los parámetros para obtener medidas similares. Por ejemplo $\alpha = 0.8, MG = 10 * n, MA = 0.0.5 * MG$:



Se ha añadido exploración al aumentar el número de vecinos generados. Se ha balanceado añadiendo convergencia reduciendo α y el número de vecinos aceptados. Se obtienen resultados similares o incluso ligeramente mejores. Por esto se ha decidido crear una nueva tabla para añadir a la comparación final:

	Iris				Ecoli				Rand				NewThyroid			
	TasaC	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T
Seed 123	0.67	0.00	0.67	23.56	22.56	71.00	24.53	102.40	0.76	0.00	0.76	23.98	13.83	6.00	14.07	103.36
Seed 456	0.67	0.00	0.67	18.81	22.00	66.00	23.83	97.12	0.76	0.00	0.76	25.03	13.83	6.00	14.07	95.98
Seed 789	0.67	0.00	0.67	20.75	22.17	80.00	24.39	98.74	0.76	0.00	0.76	26.77	13.83	6.00	14.07	100.02
Seed 101112	0.67	0.00	0.67	23.00	22.28	63.00	24.02	99.73	0.76	0.00	0.76	23.64	13.83	6.00	14.07	103.41
Seed 131415	0.67	0.00	0.67	22.76	22.59	58.00	24.20	94.53	0.76	0.00	0.76	23.53	13.83	6.00	14.07	106.96
Media	0.67	0.00	0.67	21.78	22.32	67.60	24.19	98.50	0.76	0.00	0.76	24.59	13.83	6.00	14.07	101.95

Cuadro 17: ILS-ES Geo 0.80 (10 % de restricciones)

	Iris				Ecoli				Rand				NewThyroid			
	TasaC	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T
Seed 123	0.67	0.00	0.67	23.09	21.94	164.00	24.18	101.42	0.76	0.00	0.76	26.01	14.29	0.00	14.29	103.27
Seed 456	0.67	0.00	0.67	18.99	21.82	149.00	23.85	96.97	0.76	0.00	0.76	24.12	14.29	0.00	14.29	88.37
Seed 789	0.67	0.00	0.67	24.32	22.24	138.00	24.12	106.06	0.76	0.00	0.76	26.31	14.29	0.00	14.29	100.70
Seed 101112	0.67	0.00	0.67	24.07	21.97	156.00	24.09	106.26	0.76	0.00	0.76	23.30	14.29	0.00	14.29	102.04
Seed 131415	0.67	0.00	0.67	21.70	21.89	154.00	23.99	97.97	0.76	0.00	0.76	23.89	13.83	25.00	14.30	95.14
Media	0.67	0.00	0.67	22.43	21.97	152.20	24.05	101.74	0.76	0.00	0.76	24.72	14.20	5.00	14.29	97.90

Cuadro 18: ILS-ES Geo 0.80 (20 % de restricciones)

6.2.4. Resultados de algoritmo de Comparación

Se añaden las tablas de resultados de algoritmos de comparación como referencia:

	Iris			Ecoli			Rand			NewThyroid		
	TasaC	***Inf***	**T**	***TasaC***	***Inf***	**T**	***TasaC***	***Inf***	**T**	***TasaC***	***Inf***	**T**
Seed 123	0.67	0.00	0.11	31.70	0.00	0.67	0.76	0.00	0.11	14.29	0.00	0.13
Seed 456	0.67	0.00	0.11	34.23	4.00	0.59	0.76	0.00	0.10	14.29	0.00	0.15
Seed 789	0.67	0.00	0.12	29.96	0.00	0.67	0.76	0.00	0.10	14.29	0.00	0.15
Seed 101112	0.67	0.00	0.11	33.04	0.00	0.59	0.76	0.00	0.11	14.29	0.00	0.15
Seed 131415	0.67	0.00	0.10	35.16	6.00	0.83	0.76	0.00	0.10	14.29	0.00	0.13
Media	0.67	0.00	0.11	32.82	2.00	0.67	0.76	0.00	0.11	14.29	0.00	0.14

Cuadro 19: COPKM (10 % de restricciones)

	Iris			Ecoli			Rand			NewThyroid		
	TasaC	***Inf***	**T**	***TasaC***	***Inf***	**T**	***TasaC***	***Inf***	**T**	***TasaC***	***Inf***	**T**
Seed 123	0.67	0.00	0.13	28.40	0.00	0.97	0.76	0.00	0.13	14.29	0.00	0.18
Seed 456	0.67	0.00	0.13	32.29	0.00	0.83	0.76	0.00	0.11	14.29	0.00	0.18
Seed 789	0.67	0.00	0.13	33.46	0.00	0.96	0.76	0.00	0.13	14.29	0.00	0.18
Seed 101112	0.67	0.00	0.13	28.40	0.00	0.83	0.76	0.00	0.14	14.29	0.00	0.18
Seed 131415	0.67	0.00	0.14	29.61	0.00	0.97	0.76	0.00	0.13	14.29	0.00	0.18
Media	0.67	0.00	0.13	30.43	0.00	0.91	0.76	0.00	0.13	14.29	0.00	0.18

Cuadro 20: COPKM (20 % de restricciones)

	Iris				Ecoli				Rand				NewThyroid			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	0.67	0.00	0.67	1.45	21.25	110.00	24.30	20.58	0.76	0.00	0.76	1.00	10.82	117.00	15.30	2.84
Seed 456	0.67	0.00	0.67	1.32	21.35	114.00	24.50	42.41	0.76	0.00	0.76	0.97	10.82	105.00	14.84	3.44
Seed 789	0.67	0.00	0.67	1.09	22.14	74.00	24.19	17.89	0.76	0.00	0.76	1.13	13.83	6.00	14.07	2.83
Seed 101112	0.67	0.00	0.67	1.06	22.18	66.00	24.01	22.93	0.76	0.00	0.76	1.01	10.89	95.00	14.54	2.63
Seed 131415	0.67	0.00	0.67	1.36	22.26	45.00	23.50	15.34	0.76	0.00	0.76	0.82	10.89	98.00	14.65	3.15
Media	0.67	0.00	0.67	1.26	21.84	81.80	24.10	23.83	0.76	0.00	0.76	0.99	11.45	84.20	14.68	2.98

Cuadro 21: BL (10 % de restricciones)

	Iris				Ecoli				Rand				NewThyroid			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	0.67	0.00	0.67	0.91	21.57	239.00	24.82	20.09	0.76	0.00	0.76	1.00	10.88	229.00	15.17	2.67
Seed 456	0.67	0.00	0.67	1.08	21.92	158.00	24.07	15.74	0.76	0.00	0.76	1.06	10.82	264.00	15.76	3.03
Seed 789	0.67	0.00	0.67	1.46	21.94	147.00	23.94	22.05	0.76	0.00	0.76	1.09	14.29	0.00	14.29	2.56
Seed 101112	0.67	0.00	0.67	1.15	21.80	177.00	24.21	20.55	0.76	0.00	0.76	1.06	14.29	0.00	14.29	2.70
Seed 131415	0.67	0.00	0.67	1.14	21.77	156.00	23.89	20.10	0.76	0.00	0.76	0.92	14.29	0.00	14.29	2.80
Media	0.67	0.00	0.67	1.15	21.80	175.40	24.19	19.70	0.76	0.00	0.76	1.03	12.91	98.60	14.76	2.75

Cuadro 22: BL (20 % de restricciones)

6.3. Comparación Global

Se pasa entonces a una comparación de todos los algoritmos vistos en el estudio. De esta forma se podrán distinguir los comportamientos en diferentes conjuntos:

	Iris				Ecoli				Rand				NewThyroid			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
COPKM	0.67	0.00	X	0.11	32.82	2.00	X	0.67	0.76	0.00	X	0.11	14.29	0.00	X	0.14
BL	0.67	0.00	0.67	1.26	21.84	81.80	24.10	23.83	0.76	0.00	0.76	0.99	11.45	84.20	14.68	2.98
BMB	0.67	0.00	0.67	10.26	22.84	134.00	26.55	88.83	0.76	0.00	0.76	7.98	13.83	6.00	14.07	31.43
ES Cauchy	0.67	0.00	0.67	41.03	21.75	77.00	23.88	79.55	0.76	0.00	0.76	26.71	13.24	24.40	14.18	29.95
ES Geo 0.95	0.67	0.00	0.67	25.57	21.68	82.80	23.97	92.38	0.76	0.00	0.76	22.14	13.83	6.00	14.07	94.85
ES Geo 0.98	0.67	0.00	0.67	59.46	41.60	1446.80	81.62	76.46	0.76	0.00	0.76	49.14	13.78	7.80	14.08	94.02
ILS	0.67	0.00	0.67	6.20	21.53	78.60	23.71	97.28	0.76	0.00	0.76	5.06	12.05	60.60	14.37	17.94
ILS-ES Cauchy	0.67	0.00	0.67	31.08	23.35	88.20	25.79	37.85	0.76	0.00	0.76	28.19	13.83	6.00	14.07	56.55
ILS-ES Geo 0.8	0.67	0.00	0.67	21.78	22.32	67.60	24.19	98.50	0.76	0.00	0.76	24.59	13.83	6.00	14.07	101.95

Cuadro 23: Resultados globales en el PAR con 10 % de restricciones

	Iris				Ecoli				Rand				NewThyroid			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
COPKM	0.67	0.00	X	0.13	30.43	0.00	X	0.91	0.76	0.00	X	0.13	14.29	0.00	X	0.18
BL	0.67	0.00	0.67	1.15	21.80	175.40	24.19	19.70	0.76	0.00	0.76	1.03	12.91	98.60	14.76	2.75
BMB	0.67	0.00	0.67	10.01	22.44	151.20	24.50	92.62	0.76	0.00	0.76	8.01	14.29	0.00	14.29	26.98
ES Cauchy	0.67	0.00	0.67	39.78	21.87	150.60	23.92	83.13	0.76	0.00	0.76	25.59	12.93	93.40	14.68	59.76
ES Geo 0.95	0.67	0.00	0.67	25.45	21.41	178.40	23.84	96.35	0.76	0.00	0.76	21.55	14.29	0.00	14.29	101.85
ES Geo 0.98	0.67	0.00	0.67	60.88	39.46	2515.20	73.72	79.24	0.76	0.00	0.76	48.94	14.29	0.00	14.29	96.62
ILS	0.67	0.00	0.67	6.17	21.97	119.80	23.60	101.27	0.76	0.00	0.76	4.99	12.22	138.20	14.81	18.25
ILS-ES Cauchy	0.67	0.00	0.67	30.81	22.71	182.80	25.20	39.81	0.76	0.00	0.76	28.87	14.29	0.00	14.29	79.87
ILS-ES Geo 0.8	0.67	0.00	0.67	22.43	21.97	152.20	24.05	101.74	0.76	0.00	0.76	24.72	14.20	5.00	14.29	97.90

Cuadro 24: Resultados globales en el PAR con 20 % de restricciones

De nuevo, los conjuntos más pequeños no aportan demasiada información por lo que se observa principalmente *Ecoli* y *NewThyroid*. El algoritmo COPKM tiende a reducir *Infeasibility* de forma agresiva hasta que llega a un óptimo local y para. Esto provoca que su solución sea la peor de toda la tabla. Por otro lado, la búsqueda local explora más el entorno de las soluciones alcanzando una solución bastante mejor. Aunque la búsqueda local no se centra solamente en reducir *Infeasibility* de manera tan agresiva, si que converge bastante rápido. Los resultados obtenidos por BMB son muy similares a los de la BL. Este algoritmo podría ser mejor en aquellos casos donde haya una mayor cantidad de óptimos locales distribuidos por el espacio de búsqueda. Para obtener mejores resultados se necesita sin embargo, que el conjunto pueda converger lo suficientemente rápido. A su vez el esquema de Cauchy obtiene resultados muy similares a un descenso geométrico para el enfriamiento simulado. Aunque en el conjunto NewThyroid se observa como Cauchy tiende a converger más rápido y por tanto atascarse en óptimos locales. Por último, las búsquedas iterativas muestran un comportamiento similar a la BMB. En conjuntos pequeños donde se converge rápidamente a la solución no han aportado nada. Sin embargo, en conjuntos grandes la cantidad de evaluaciones necesarias para terminar de converger supera a las permitidas. Esto provoca que se necesiten aplicar estrategias de convergencia más fuertes (se estancan más en óptimos loca-

les). Por último cabe destacar la diferencia entre las velocidades de enfriamiento usadas. Mientras que el esquema de Cauchy ha proporcionado buenos resultados en ES y ILS-ES, se ha demostrado que la variación de diferentes parámetros puede permitir llegar a mejores soluciones de forma general. Sin embargo, el hallar estos parámetros puede ser costoso y a veces inviable. Se trata de aplicar el algoritmo específico según las características del problema y las posibles condiciones de este. En problemas con espacios de búsqueda simples BL o incluso COPKM pueden ser algoritmos viables si prima la velocidad. Por otro lado, se ha hablado de que si sigue primando la velocidad pero el espacio de búsqueda tiene muchos óptimos locales, algoritmos como BMB o ILS pueden ser indicados. Por último, en espacios de búsqueda mas complejos o si se poseen los recursos necesarios y la intención de encontrar las mejores soluciones ES o ILS-ES pueden ser convenientes gracias a su capacidad de exploración manteniendo cierta convergencia.

6.4. Conclusión

Como conclusión del estudio, se puede decir que la elección de los algoritmos depende mucho de la situación. Resultados relativamente óptimos se pueden conseguir en una pequeña cantidad de tiempo gracias a la búsqueda local o greedy. La BMB o ILS puede ser útil si los resultados de BMB varían mucho dependiendo de la solución inicial. Aunque se ha visto que pueden no ser convenientes si no tienen las suficientes evaluaciones a su disposición. Por último ES se aplica a casos más complejos. Se ha demostrado que el algoritmo ES contiene muchos parámetros que pueden modificar de forma radical su comportamiento. Si se combina con ILS, se añade mayor complejidad a su comportamiento pero a su vez, una mayor capacidad de adaptación dependiendo del problema. Se ha propuesto un ejemplo práctico de como modificar alguno de sus parámetros con restricción en del esquema de enfriamiento a usar. Aunque no se hayan analizado a profundidad todos los componentes, se demuestra que cualquiera de ellos provoca un gran impacto en el comportamiento del algoritmo.

Referencias

[1] TFM: Clustering de documentos con restricciones de tamaño
https://pdfs.semanticscholar.org/812b/edf1c055c37d03b5a2acd655fead801bd215.pdf

[2] Constrained K-means Clustering with Background Knowledge
https://www.cs.cmu.edu/~./dgovinda/pdf/icml-2001.pdf