UNIVERSIDAD
DE GRANADA

METAHEURÍSTICAS

GRADO EN INGENIERÍA INFORMÁTICA

PRÁCTICA 2

TÉCNICAS DE BÚSQUEDA BASADAS EN POBLACIONES 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.

Se comenzará explicando las estructuras de datos utilizadas. Se ha utilizado una matriz X de $n \times d$ para guardar los datos. Se podría haber optado por la estrategia de la anterior práctica y añadir las columnas necesarias a la matriz para el calculo de los parámetros. Sin embargo, esto significaría un gran aumento de la memoria utilizada para una mejora de eficiencia mínima (todas las matrices tendrían d columnas iguales). Por ello se opta en mantener todos los parámetros en diferentes vectores. Gracias a la previa factorización de las funciones se han podido adaptar de forma fácil. Cada vector a su vez será parte de un vector de N elementos (tamaño de la población). Es decir, el elemento i de la población podrá acceder a todos sus parámetros en la posición i de los diferentes vectores. La estructura a la que accedan será diferente dependiendo del parámetro, desde número para el valor de la función objetivo, hasta un vector para las posiciones de los diferentes centroides.

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 poblaciones

Para la creación de la población inicial se ha optado por un método sencillo:

Algorithm 7: Generar Población Inicial

```
1 GeneratePopulation (N, n, k)
      input : Tamaño Población N, Número instancias n, Número clases k
      output: Vector de soluciones Clusters
2     Clusters ← [N, n]
3     for i ← 0 to N − 1 do
4         Clusters_i ← RandomInRange(0, k − 1, n)
5         CheckEmptyCluster(Clusters_i)
6     return Clusters
```

Se aprecia que es una función sencilla y eficiente. Se crea una matriz para guardar N soluciones compuestas por n asignaciones. La función $RandomInRange()$ permite crear números aleatorios en un rango en especifico, en este caso $[0, k-1]$. El tercer argumento se refiere a cuantos números aleatorios en este rango debe devolver, en este caso n. Con esta llamada, se asigna un *cluster* a cada instancia para un solo elemento de la población. Se repite la llamada para rellenar todos los elementos de la población. Luego de cada llamada a $RandomInRange()$, se comprueba que cumple la restricción de no dejar ningún *cluster* vacío. Dado que la función devuelve números sobre una distribución uniforme discreta esto no debería pasar pero para asegurar un buen funcionamiento en casos extremos, se realizan estos chequeos.

Uno de los procesos comunes en los algoritmos será el proceso de selección. Se ha elegido un torneo binario para esto:

Algorithm 8: Selección

```
1 Selection (N, P)
      input : Tamaño Población N, Número de Seleccionados P
      output: Vector de seleccionados Selected
2     Selected ← [P]
3     for i ← 0 to P − 1 do
4         posible ← RandomInRange(0, N − 1, 2)
5         if PopulationObjetive_posible[0] < PopulationObjetive_posible[1] then
6             Selected_i ← posible[0]
7         else
8             Selected_i ← posible[1]
9     return Selected
```

Es un algoritmo bastante sencillo. Se devuelve un vector con los índices de los padres seleccionados. Este vector será rellenado uno a uno mediante un torneo binario. Con *RandomInRange* se obtienen dos índices aleatorios de la población. Se realiza el 'torneo' comparando el valor de la función objetivo de ambos e insertando el mejor de ellos.

El siguiente proceso común en los algoritmos es el cruce. Para este proceso se utiliza la esperanza matemática en el cálculo del número esperado de hijos: $N_C = P_{cross} * N$. Este número de hijos será calculado previamente. Debido a que cada pareja produce dos hijos, se redondea el número de hijos al número par superior. Por cada pareja de hijos se deberían escoger dos padres de forma aleatoria entre los seleccionados. Para evitar cálculos innecesarios, se aprovecha la aleatoriedad que ya aplica el mecanismo de selección y se cruzan las $N_C/2$ primeras parejas de la población intermedia. Se obtiene el siguiente algoritmo:

Algorithm 9: Generate Offspring

```
1 GenerateOffspring (N_C, S)
    input : Número de Hijos N_C, Vector de seleccionados Selected
    output: Vector de Hijos H
2   H ← [N_C]
3   i ← 0
4   while i ≠ N_C do
5     H[i] ←
        CrossOver(PopuCluster_Selected[i], PopuCluster_Selected[i+1])
6     H[i + 1] ←
        CrossOver(PopuCluster_Selected[i], PopuCluster_Selected[i+1])
7     i ← i + 2
8   return H
```

Para este algoritmo primero se crea el vector H con el espacio necesario para los hijos. Se realizan entonces $N_C/2$ iteraciones el bucle (se ha explicado previamente que N_C es par). Por cada una de estas iteraciones, se han seleccionado dos padres de la población intermedia. Se utiliza la función *CrossOver* para calcular el hijo resultante de los dos padres. Esta función dependerá de los argumentos especificados por el usuario al inicio dele programa, y puede redirigir al operador de cruce uniforme o al operador de cruce por segmento fijo. Se describen estos a continuación.

Al inicio del programa el usuario puede elegir que operador de cruce sera usado

en el proceso. Se comienza describiendo el operador de cruce uniforme:

Algorithm 10: Uniform CrossOver

```
1 UniformCrossOver (P1, P2)
      input : Vector de Clusters P1, Vector de Clusters P2
      output: Hijo C
2     C ← P1
3     indexCopy ← RandomSample(0, n, n/2)
4     foreach i ∈ indexCopy do
5         C[i] ← P2[i]
6     return C
```

El operador de cruce uniforme produce un hijo C en base a los padres $P1$ y $P2$ combinando de manera uniforme las características de los dos. En vez de generar n números aleatorios para elegir de que padre copiar la característica, se generan $n/2$ que indicaran los índices de las características a copiar. A pesar de que parezca contraproducente, la forma mas eficiente de realizar esto es copiando uno de los padres en el hijo y luego sobrescribir las características seleccionadas. Esto se debe a que los índices de las características no se obtienen de forma ordenada. Por esto, si no se quisiese sobrescribir, se deberían realizar búsquedas sobre $indexCopy$ para cada posición en n. Con este procedimiento se obtiene un hijo la mitad de las características de cada padre.

También se puede escoger el operador de cruce por segmento fijo. Este operador produce un hijo seleccionando de un padre un segmento continuo de características y copiándolo sin modificación a la descendencia. Este segmento se genera aleatoriamente en cada cruce para dar aleatoriedad. Además, el resto de genes se combinan de manera uniforme con ambos padres.

Algorithm 11: Two Point Uniform CrossOver

```
1 TwoPointUniformCrossOver (P1, P2, BestFirst)
      input : Vector de Clusters P1, Vector de Clusters P2, Booleano
              BestFirst
      output: Hijo C
2     P1, P2 ← chooseOrder(P1, P2, BestFirst)
3     C ← P1
4     start, length ← RandomInRange(0, n − 1, 2)
5     end ← (start + length) %n
6     shared ← n − length
7     C ← UniformCrossOverRange(P1, P2, C, end, shared)
8     return C
```

En este caso, se introduce el parámetro $BestFirst$ en la función de cruce. Ya que este operador favorece uno de los padres, se da la posibilidad de que el usuario escoja que padre favorecer. Si se escoge el mejor padre se favorece la explotación (mayor convergencia). Si se escoge el peor padre se favorece la exploración (menor convergencia). La función *choseOrder* ordenará los padres de forma acorde a este parámetro. Por los motivos descritos anteriormente, se decide copiar el padre en el vector hijo. Se pasa a calcular *end*, el cual se refiere al final del segmento fijo copiado del padre favorecido y el comienzo del segmento compartido por ambos padres. Tras esto *shared* refleja la longitud del segmento compartido por ambos padres. La función $UniformCrossOverRange()$ replica el comportamiento del operador de cruce uniforme en el el segmento especificado por *end* y *shared* (se contempla el caso de que el segmento este formado por dos segmento al principio y al final).

Estos cruces pueden producir hijos no factibles ya que alguno de los *clusters* puede estar vacío. Por cada uno de estos hijos creados a partir de cruces, se realiza un conteo en cada uno de los *clusters* para comprobar esto. Si alguno de los *clusters* están vacíos se asignarán instancias pertenecientes a *clusters* que contengan mas de una. Estos datos de conteo también servirán para el siguiente operador de mutación:

Algorithm 12: Mutate Children

```
1 MutateChildren (C)
      input  : Hijo C
      output: Hijo C
2     GenIndex ← RandomInRange(0, n − 1)
3     NewC ← RandomInRange(0, k − 1)
4     while OneInstCluster(GenIndex) do
5         GenIndex ← GenIndex + 1
6     if SameCluster(GenIndex, NewC) then
7         NewC ← NewC + 1
8     C[GenIndex] ← NewC
9     return C
```

El operador de mutación realiza una mutación aleatoria en un hijo especificado. Para esto escoge un gen de forma aleatoria y lo cambia a un *cluster* diferente al suyo. Antes de cambiarlo primero comprueba que el *cluster* del gen a mutar tiene más de un elemento (se necesita respetar las restricciones). Itera (de forma circular) por el vector hasta encontrar un gen apropiado. Tras esto, comprueba que la nueva clase sea diferente de la actual (se fuerza la mutación). Se cambia la asignación del gen al nuevo *cluster* y se devuelve el hijo.

Con esto, se terminan las consideraciones comunes de los algoritmos usados.

Se pasa entonces a explicar cada algoritmo en mayor profundidad.

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 13: 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 14: 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 15: 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 16: 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.

3. Algoritmos de Búsqueda basados en Poblaciones

3.1. Algoritmos genéticos estacionarios

Una de las heurísticas implementadas en esta práctica son los AGE(algoritmos genéticos estacionarios). En este algoritmo se parte de una población inicial de 50 cromosomas. Cada generación genera un total de 2 hijos que podrán ser o

no reinsertados en la población si se consideran mejores cromosomas que los dos peores de la población. Se seleccionan dos padres mediante un torneo binario y estos siempre se cruzarán para proporcionar los dos hijos. Estos hijos tendrán una probabilidad $P_m = 0.001$ de mutar cualquiera de sus genes. Que hijos mutar se puede calcular de la siguiente forma:

Algorithm 17: Mutate Steady Children

```
1  MutateSteadyChildren (H)
       input  : Vector de Hijos H
       output: Vector de Hijos H
2      SteadyLimit ← P_m * n foreach C ∈ H do
3          if RandomInRange(0, 1) < SteadyLimit then
4              C ← MutateChildren(C)
5      return H
```

Como se puede apreciar, se ha usado la esperanza matemática para calcular si cada cromosoma va a sufrir una mutación. Al ser una probabilidad tan baja, este cambio aumenta la eficiencia al reducir la cantidad de números aleatorios de manera significativa. Se describe a continuación el algoritmo AGE:

Algorithm 18: Steady State Genetic Algorithm

```
1  SteadyState (X, R, N)
       input  : Matriz de datos X, Matriz de restricciones R, Tamaño de
                población N
       output: Población P
2      population ← GenerateInitialPopulation(X, R, N)
3      evaluations ← N
4      while evaluations < 100000 do
5          evaluations ← evaluations + 2
6          parents ← SelectParents(P, 2)
7          for Pair P1, P2 ∈ parents do
8              H ← H + CrossOver(P1, P2) H ← H + CrossOver(P1, P2)
9          FixChildren(H)
10         H ← MutateSteadyChildren(H)
11         H ← CalculateFitness(H)
12         P ← ReinsertSteadyChildren(P, H)
13     return P
```

Se aprecia que es un algoritmo sencillo y directo pero se comentará brevemente. Primero se creara la población inicial a partir del algoritmo generador de soluciones aleatorias descrito anteriormente. El criterio de parada de este algoritmo serán

100000 evaluaciones de la función objetivo. Al crear la población inicial se han evaluado N funciones objetivo. Este algoritmo genera dos nuevos cromosomas por cada generación (aunque se podría cambiar). Esto significa que se evalúan dos funciones objetivo por cada generación. En cada una seleccionaremos dos padres mediante el torneo binario. Cada pareja de padres en *parents* produce 2 hijos. Como ya se comentó, estos hijos generados pueden no ser factibles por lo que se puede necesitar arreglarlos para que cumplan las restricciones necesarias. Para comprobar las restricciones, hacemos un conteo del número de instancias en cada *cluster*. Este conteo también servirá para la operación de mutación de los hijos. Por último, se calcula la función objetivo para los hijos y se intenta reinsertarlos en la población original. Los hijos sustituirán a los dos peores elementos de la población si y solo si son mejores que ellos (comparando función objetivo).

3.2. Algoritmos genéticos generacionales

La siguiente heurística implementada se trata de los AGG(algoritmos genéticos generacionales). Esta versión sigue un esquema muy parecido al anterior pero con tres notables diferencias. La primera se trata del número de hijos (y padres), el cuál sera del mismo tamaño que la población. Esto implica una menor cantidad de generaciones ya que el número de evaluaciones por generación crece. La segunda diferencia se trata de la introducción de una probabilidad de cruce $P_C = 0.7$. No todas los padres se cruzarán y algunos pasarán a la población de hijos de manera directa. La última diferencia se trata del método de reinserción. No se compite por insertar, sino que todos los hijos sustituyen a la población. Sin embargo, se sigue un esquema **elitista**, por lo que la mejor solución de la población siempre se mantendrá.

También se añade una pequeña diferencia en la forma de calcular las mutaciones. En el algoritmo AGE se genera un número aleatorio para cada hijo. En este caso, se puede reducir el número de generaciones todavía más. Para esto se puede calcular el número de mutaciones esperadas en los hijos y a partir de este número,

generar solo los números aleatorios que necesitemos:

Algorithm 19: Mutate Generational Children

```
1  MutateGenerationalChildren (H)
       input  : Vector de Hijos H
       output: Vector de Hijos H
2      Mutations_N ← P_m * n * N
3      RandomNumbers ← RandomInRange(0, N * n, Mutations_N)
4      foreach r ∈ RandomNumbers do
5          IndexCrom = r//n
6          IndexGen = r %n
7          H ← MutateChildren(H, IndexCrom, IndexGen)
8          if NeedToRecalculate(IndexCrom) then
9              Recalculate(IndexCrom)
10     return H
```

Es un algoritmo sencillo que hace uso de la esperanza matemática para calcular el número de mutaciones que deberían aparecer en la población. Si se generan números aleatorios en el rango $[0, N * n]$, se puede ahorrar un mayor número de generaciones de números ya que cada número generado especifica un cromosoma y un gen específico a mutar. Se ha intentado optimizar lo máximo posible los cálculos del programa. Por ello, si los parámetros han sido calculados en una generación pasada, solo se necesitará copiarlos en vez de volver a calcularlos para los hijos que se han mantenido igual. Sin embargo, la mutación puede afectar a nuevos hijos(producto de cruce) o aquellos que pasaron de forma directa. La función $NeedToRecalculate()$, nos devolverá si el cromosoma mutado es producto de un cruce (aún no se han calculado sus parámetros) o ha pasado de forma directa(se necesitan recalcular los parámetros). Se pasa entonces a la descripción general del

algoritmo:

Algorithm 20: Generational Genetic Algorithm

```
1  Generational (X, R, N)
       input  : Matriz de datos X, Matriz de restricciones R, Tamaño de
                población N
       output: Población P
2      population ← GenerateInitialPopulation(X, R, N) evaluations ← N
3      while evaluations < 100000 do
4          evaluations ← evaluations + N
5          parents ← SelectParents(P, N)
6          parents_N ← RoundEven(0.7 * N)
7          bestUsed ← WillBestOnePass(parents)
8          for PairP1, P2 ∈ parents[0..parents_N] do
9              H ← H + CrossOver(P1, P2)
10             H ← H + CrossOver(P1, P2)
11         FixChildren(H)
12         H ← CopyRestParents(H, parents_H)
13         H ← MutateGenerationalChildren(H)
14         H ← CalculateFitness(H)
15         bestUsed ← checkIfBestMuted(H)
16         P ← ReinsertSteadyChildren(P, H, bestUsed)
17     return P
```

Se comienza de la misma forma creando una población inicial a partir del algoritmo de generación de soluciones aleatorias. Se establece una condición de parada y se entra en el bucle principal. En esta ocasión, cada generación añade N evaluaciones de la función objetivo. Se escogen N padres para crear la población intermedia. Se calcula el número de padres que se utilizarán para la generación de hijos. Se añade un booleano *bestUsed*, que indicará si la mejor solución de la población formará parte de los hijos (no es cruzada). Se puede realizar esta función ya que se sabe que padres se escogen para cruzar(los $parents_N$ primeros). Tras esto, se generan los hijos necesarios y se arreglan si no cumplen las restricciones. Se copia el resto de padres al vector de hijos de forma directa y se mutan los hijos. Se calcula el valor de la función objetivo para los hijos (solo los nuevos o quien haya pasado directamente pero haya sido mutado) y se comprueba si el índice del cromosoma mutado es el del mejor cromosoma de la población. Por último se sustituye la población por los hijos manteniendo el elitismo. En este caso, *bestUsed* indica si el mejor cromosoma de la población ha pasado intacto a la siguiente generación. Si es así, se puede sustituir la población directamente. Si no, el peor cromosoma de los hijos será reemplazado por el mejor cromosoma de la población para mantener el elitismo.

3.3. Algoritmos meméticos

La última metaheurística implementada son los AM(algoritmos meméticos). Los AM son hibridaciones entre algoritmos evolutivos y algoritmos de búsqueda local. Los algoritmos evolutivos son buenos exploradores y malos explotadores mientras que los algoritmos de búsqueda local son lo contrario. Con esto se pretende mejorar los resultados y reducir el tiempo. Se utiliza el algoritmo genético generacional con la búsqueda local utilizada anteriormente. Se reduce la población a 10 cromosomas y se mantiene la probabilidad de cruce de $P_C = 0.7$ y de mutación de $P_m = 0.001$. La búsqueda local se aplica cada 10 generaciones. La posibilidad de aplicar la búsqueda local permite diferentes tipos de variaciones. Se exploran 3 posibilidades de hibridación:

1. Aplicar BLS sobre todos los cromosomas de la población
2. Aplicar BLS sobre un subconjunto de la población aleatorio $P = 0.1$
3. Aplicar BLS sobre los $0.1 * N$ mejores cromosomas de la población

Para evitar que se desperdicien evaluaciones de la BLS en soluciones de calidad se añade un criterio de parada solo a esta. En cada cromosoma al que se pretenda aplicar BLS, se admitirán solo ξ fallos, siendo $\xi = 0.1 * n$. Se dice que falla si no produce cambios en el cromosoma. De esta manera, si el cromosoma no es optimizable, el contador de fallos crecerá rápidamente y la BL se detendrá. Se

pasa a describir la BL utilizada:

Algorithm 21: Búsqueda Local Débil

```
1  LocalSearchSoft (P, Index)
       input  : Parámetros del cromosoma P, Índice Index
       output: Parámetros del cromosoma P
2      BestValues ← CopyValues(···)
3      OriginalCluster ← GetActualCluster(X, Index)
4      foreach PossibleCluster ∈ k do
5          OldCluster ← GetActualCluster(X, Index)
6          if OriginalCluster == PossibleCluster then
7              Continue
8          M, SumValuesCluster ← CalculateCentroidesOptimizado(···)
9          X, SumDist, OldDist ← CalcularDistanciasOptimizado(···)
10         ObjectiveValue = CalcularObjectiveValue(···)
11         if ObjectiveValue ≤ BestObjectiveValue then
12             BestValues ← CopyValues(···)
13     Changed ← BestCluster == OriginalCluster
14     if Changed then
15         P ← BestValues
16     return P, Changed
```

Esta función realizará una búsqueda local dado todos los parámetros del cromosoma y el índice del gen en cuestión. Primero se copiaran los valores de los parámetros en vectores adicionales y se guarda el *cluster* original. Se entra entonces en un bucle que prueba con cada uno de los *cluster* posibles. Cuidando que sea diferente del original(se comprueba fuera que no se dejará vacío). Gracias a la implementación anterior de la BL, las funciones permiten la factorización de los cálculos por lo que se puede calcular los nuevos parámetros de forma rápida. Se compara los valores de la función objetivo y si estos son mejores que los guardados, se conservan. Tras este bucle se comprueba si el mejor *cluster* fue el original. Si no, significa que existe la posibilidad de una mejora. Actualizamos los parámetros con los mejores y se devuelven. También se devuelve *Changed* para el conteo de fallos.

Con esto se explica la mayor parte del programa. Sin embargo se expone el

código general de la función:

Algorithm 22: Memetic Algorithm

```
1  Memetic (X, R, N)
       input  : Matriz de datos X, Matriz de restricciones R, Tamaño de
                población N
       output: Población P
2      population ← GenerateInitialPopulation(X, R, N) evaluations ← N
3      while evaluations < 100000 do
4          .
5          Same algorithm as AGG
6          .
7          generations ← generations + 1
8          if generations == Interval then
9              generations ← 0
10             selectedBLS ← SelectBLS(P)
11             foreach x ∈ selectedBLS && ξ < ErrorLimit do
12                 ξ ← 0
13                 if P[x]_COUNT ! = 1 then
14                     GenOrder ← RandomOrder(n)
15                     foreach I ∈ GenOrder && ξ < ErrorLimit do
16                         evaluations ← evaluations + (k − 1)
17                         P[x], Changed ← LocalSearchSoft(P[x], I)
18                         if ! Changed then
19                             ξ ← ξ + 1
20     return P
```

Como se puede ver la mayor parte del algoritmo se comparte con AGG(se suprime por claridad). Se añade una variable *generations* que cuenta las generaciones creadas. Dependiendo del intervalo que haya sido introducido por el usuario, la BLS se activará más o menos frecuente. Cuando se active, se resetea el contador a 0 y se crea una lista de los cromosomas a los que se van a aplicar la BL (una de las tres variaciones ya discutidas). Por cada uno de estos cromosomas, primero se comprobará que no dejará el *cluster* vacío al modificarlo. Tras esto, se crea una orden aleatorio en el que explorar todos los genes. Sabiendo el cromosoma y el gen a explorar podemos utilizar la función ya explicada antes. Esta nos dirá si ha habido alguna modificación o no en el cromosoma, con lo que se actualiza el conteo de errores de la forma adecuada.

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 dos programas **main.py** y **memetic.py**. El primero se corresponde a los algoritmos genéticos y el segundo a los algoritmos meméticos. 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 realizar las comparaciones. Tanto **main.py** como **memetic.py** escriben los resultados en ficheros .csv para su posterior análisis. Esto provoca un pequeño *overhead* que puede ser despreciado debido al tiempo que total que toman los programas en ejecutarse.

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
```

El programa **main.py** se puede lanzar con o sin argumentos. Al lanzarse sin argumentos se realiza una ejecución por defecto. Para especificar una ejecución se lanza con los siguientes 9 argumentos:

```
$ python3 main.py [model] [dataset]
                  [Restr.%] [Seed] [LambdaMod]
                  [N_Population] [Mut_Prob]
                  [Uniform] [BestFirst]
```

- **model**: modelo escogido $\in$ { **steady**, **generational**}

- **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)
- **N_Population**: tamaño de la población $\in \mathbb{N}$
- **Mut_Prob**: probabilidad de mutación $\in \mathbb{Q}$
- **Uniform**: operador de cruce uniforme $\in$ {**si**, **no**}
- **BestFirst**: si operador de cruce por segmento uniforme, el mejor padre es el primero $\in$ {**si**, **no**}

Un ejemplo en una ejecución normal sería:

```
$ python3 main.py steady ecoli 10 123 1 50 0.001 no si
```

Para la ejecución del programa **memetic.py** se modifica el parámetro **model** y se añaden 2 parámetros más.

- **model**: modelo escogido $\in$ { **best**, **random**, **all**}
- **Interval**: intervalo de generaciones entre las que se aplica la búsqueda local $\in \mathbb{N}$
- **Subset**: si modelo escogido es **best**: tamaño del subconjunto de los mejores con respecto al original. Si modelo escogido es **random**: probabilidad de ser seleccionado para subconjunto al que aplicar la búsqueda local. $\in \mathbb{Q}$

Se obtendría entonces:

```
$ python3 memetic.py best ecoli 10 123 1 50 0.001 no si 10 0.1
```

Por otra parte, se puede comentar brevemente **comparison.py**. Será utilizado para iterar ejecuciones de **main.py** y **memetic.py**, y escribirlas en archivos. 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. Optimización

Se realizan pequeñas optimizaciones con respecto a las versiones anteriores. La capacidad de modularización se ha llevado al máximo. Se realiza un gasto de memoria manteniendo todos los valores de los parámetros para poder hacer el estudio del problema en todos los casos. Potencialmente se podría reducir este gasto de memoria y potencialmente algo de tiempo guardando solo los valores necesarios entre las iteraciones (función objetivo). Si quisiera haber hecho las medidas sin este *overhead* pero se habría tardado demasiado. Haciendo un estudio de *profiling* en 4 casos se descubre que este *overhead* no llega al 5 % del tiempo total.

Este estudio de *profiling* desvela que la mayor parte del tiempo (60 %-70 %) se gasta en el cálculo de *infeasibility*. Se intenta optimizar con la ayuda de *Cython* (llamando funciones de C a través de Python) y se consigue una mejora de 5 %-10 %. Sin embargo se ha descartado por la dificultad extra que conlleva (compilación,

paso de parámetros, librerías especializadas...). Se consigue sin embargo una gran optimización de el cálculo de *infeasibility* guardando punteros a las restricciones de cada fila. No supone una gran carga en el sistema por el pequeño tamaño de los datos pero puede no ser apropiado para conjuntos de datos mayores. Se consigue reducir casi a la mitad el tiempo consumido para calcular *infeasibility*.

	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	72.46	26.04	118.00	29.31	39.98	0.72	0.00	0.72	69.21	10.81	98.00	14.57	92.94
Seed 456	0.67	0.00	0.67	72.56	24.29	177.00	29.19	37.95	0.72	0.00	0.72	68.91	10.79	95.00	14.43	91.21
Seed 789	0.67	0.00	0.67	73.87	21.15	192.00	26.46	37.99	0.72	0.00	0.72	68.92	13.48	11.00	13.91	91.65
Seed 101112	0.67	0.00	0.67	72.78	20.31	226.00	26.56	36.69	0.72	0.00	0.72	70.34	13.83	234.0	13.90	91.17
Seed 131415	0.67	0.00	0.67	72.00	21.06	105.00	23.96	38.55	0.72	0.00	0.72	68.69	10.87	98.00	14.63	95.47
Media	0.67	0.00	0.67	72.73	22.57	163.60	27.09	38.23	0.72	0.00	0.72	69.21	11.96	75.50	14.29	92.49

6.3. 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. Como se ha descrito anteriormente, las medidas temporales tienen cierto *overhead* resultado de escribir los datos en ficheros para su posterior análisis. Principalmente se compara las diferentes heurísticas basadas en poblaciones con la heurística de búsqueda local y el algoritmo de comparación. Se dejará una comparación más profunda entre las diferentes heurísticas de poblaciones para futuros trabajos. Sin embargo, se realizan comparaciones superficiales sobre todo a nivel de convergencia con gráficas.

6.3.1. Análisis de algoritmos genéticos

En este apartado se muestran los principales datos obtenidos a partir de los algoritmos genéticos. Los algoritmos genéticos han seguido dos esquemas de evolución. El esquema estacionario (AGE) y el esquema generacional con elitismo (AGG). Cada esquema se ha probado con 3 operadores de cruce diferentes. Un operador de cruce uniforme y dos operador de cruce de segmento uniforme. Se decide probar con ambos operadores de segmento uniforme para poder analizar los datos en un conjunto mayor y confirmar las diferentes hipótesis.

6.3.1.1. Algoritmo genéticos estacionarios

Se comenzará con los algoritmos genéticos estacionarios. Se podría deducir que estos algoritmos van a converger de una forma algo más lenta (con respecto al número de generaciones) que los AGG. Por ello puede que se favorezca de operadores de cruce que fuercen a converger de una forma algo más rápida. 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	0.63	6.00	0.67	244.18	21.77	84.00	24.09	844.33	0.67	6.00	0.72	218.35	13.57	13.00	14.07	401.58
Seed 456	0.64	4.00	0.67	251.75	21.96	73.00	23.98	814.45	0.67	6.00	0.72	224.88	10.66	104.00	14.65	408.86
Seed 789	0.64	4.00	0.67	243.57	22.01	70.00	23.94	806.27	0.68	4.00	0.72	235.09	10.40	110.00	14.62	413.71
Seed 101112	0.67	0.00	0.67	246.98	21.84	69.00	23.75	873.44	0.72	0.00	0.72	224.94	10.81	113.00	15.14	397.15
Seed 131415	0.67	0.00	0.67	240.34	21.33	110.00	24.37	814.04	0.72	0.00	0.72	220.91	13.60	12.00	14.07	415.13
Media	0.65	2.80	0.67	245.36	21.78	81.20	24.03	830.51	0.69	3.20	0.72	224.84	11.81	70.40	14.51	407.29

Cuadro 1: AGE (10 % de restricciones) OpCruce: Uniform

	Iris				Ecoli				Rand				NewThyroid			
	TasaC	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T
Seed 123	0.64	9.00	0.67	325.85	24.50	178.00	26.92	1222.85	0.67	12.00	0.72	301.98	13.88	22.00	14.29	554.75
Seed 456	0.63	11.00	0.67	328.47	21.89	170.00	24.20	1193.47	0.67	11.00	0.72	299.19	10.62	275.00	15.76	565.98
Seed 789	0.65	7.00	0.67	323.59	21.71	169.00	24.01	1187.64	0.67	11.00	0.72	310.78	14.04	13.00	14.29	579.46
Seed 101112	0.67	0.00	0.67	318.77	24.73	165.00	26.97	1209.54	0.72	0.00	0.72	297.10	14.29	0.00	14.29	559.46
Seed 131415	0.67	0.00	0.67	325.57	21.71	148.00	23.73	1254.46	0.72	0.00	0.72	299.07	10.54	249.00	15.20	570.59
Media	0.65	5.40	0.67	324.45	22.91	166.00	25.17	1213.59	0.69	6.80	0.72	301.63	12.67	111.80	14.76	566.05

Cuadro 2: AGE (20 % de restricciones) OpCruce: Uniform

	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	244.27	21.91	81.00	24.15	840.06	0.72	0.00	0.72	230.12	13.83	6.00	14.07	412.62
Seed 456	0.67	0.00	0.67	241.99	22.16	74.00	24.21	805.50	0.72	0.00	0.72	224.75	13.83	6.00	14.07	426.50
Seed 789	0.63	6.00	0.67	246.65	22.22	75.00	24.29	805.81	0.68	5.00	0.72	230.19	10.81	114.00	15.18	411.00
Seed 101112	0.67	0.00	0.67	245.86	22.40	69.00	24.31	805.47	0.72	0.00	0.72	229.93	10.82	115.00	15.23	416.03
Seed 131415	0.64	5.00	0.67	245.06	21.31	94.00	23.91	818.28	0.69	3.00	0.72	222.18	13.45	16.00	14.07	410.94
Media	0.65	2.20	0.67	244.76	22.00	78.60	24.17	815.03	0.70	1.60	0.72	227.43	12.55	51.40	14.52	415.42

Cuadro 3: AGE (10 % de restricciones) OpCruce: BestFirst

	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	321.10	21.90	177.00	24.32	1198.71	0.72	0.00	0.72	297.56	14.29	0.00	14.29	560.83
Seed 456	0.67	0.00	0.67	322.50	21.77	178.00	24.19	1245.36	0.72	0.00	0.72	299.91	14.29	0.00	14.29	566.18
Seed 789	0.65	5.00	0.67	327.05	21.88	176.00	24.28	1293.79	0.69	8.00	0.72	297.00	14.29	0.00	14.29	575.58
Seed 101112	0.67	0.00	0.67	318.65	21.70	200.00	24.43	1190.62	0.72	0.00	0.72	301.75	10.89	233.00	15.26	566.10
Seed 131415	0.63	13.00	0.67	325.40	21.85	155.50	23.96	1183.17	0.67	11.00	0.72	306.29	14.24	2.50	14.29	555.72
Media	0.66	3.60	0.67	322.94	21.82	177.30	24.23	1222.33	0.70	3.80	0.72	300.50	13.60	47.10	14.48	564.88

Cuadro 4: AGE (20 % de restricciones) OpCruce: BestFirst

	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	248.59	23.56	146.00	23.56	806.54	0.72	0.00	0.72	227.00	15.20	114.00	15.20	408.17
Seed 456	0.67	0.00	0.67	244.69	24.06	63.00	24.06	834.46	0.72	0.00	0.72	223.75	14.07	6.00	14.07	408.09
Seed 789	0.67	6.00	0.67	251.30	23.95	67.00	23.95	827.21	0.72	4.00	0.72	230.17	14.07	6.00	14.07	406.42
Seed 101112	0.67	0.00	0.67	245.23	24.49	119.00	24.49	811.11	0.72	0.00	0.72	222.88	14.07	6.00	14.07	405.60
Seed 131415	0.67	5.00	0.67	249.33	24.64	101.00	24.64	812.07	0.72	3.00	0.72	228.77	14.76	107.00	14.76	408.49
Media	0.67	2.20	0.67	247.83	24.14	99.20	24.14	818.28	0.72	1.40	0.72	226.51	14.43	47.80	14.43	407.35

Cuadro 5: AGE (10 % de restricciones) OpCruce: BestLast

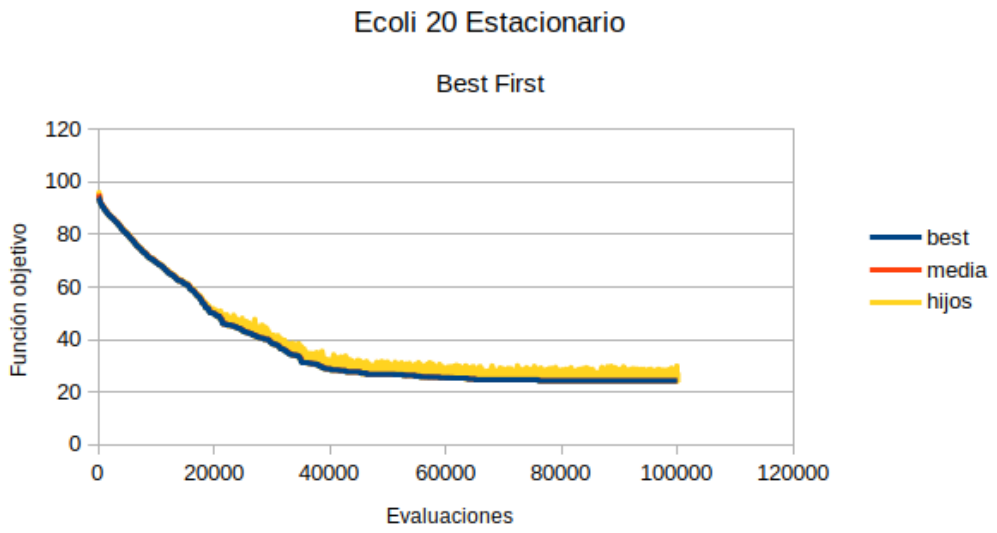
	Iris				Ecoli				Rand				NewThyroid			
	TasaC	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T
Seed 123	0.67	0.00	0.67	325.15	24.54	168.00	24.54	1181.81	0.72	0.00	0.72	295.99	14.29	0.00	14.29	576.10
Seed 456	0.67	0.00	0.67	325.37	24.29	175.00	24.29	1189.74	0.72	0.00	0.72	306.98	14.29	0.00	14.29	559.83
Seed 789	0.67	6.00	0.67	330.86	24.44	179.00	24.44	1189.07	0.72	8.00	0.72	305.66	15.67	259.00	15.67	567.28
Seed 101112	0.67	0.00	0.67	321.20	24.96	231.00	24.96	1200.75	0.72	0.00	0.72	299.06	15.76	264.00	15.76	561.19
Seed 131415	0.67	13.00	0.67	326.92	24.68	189.00	24.68	1189.28	0.72	9.00	0.72	299.87	14.29	3.00	14.29	560.73
Media	0.67	3.80	0.67	325.90	24.58	188.40	24.58	1190.13	0.72	3.40	0.72	301.51	14.86	105.20	14.86	565.03

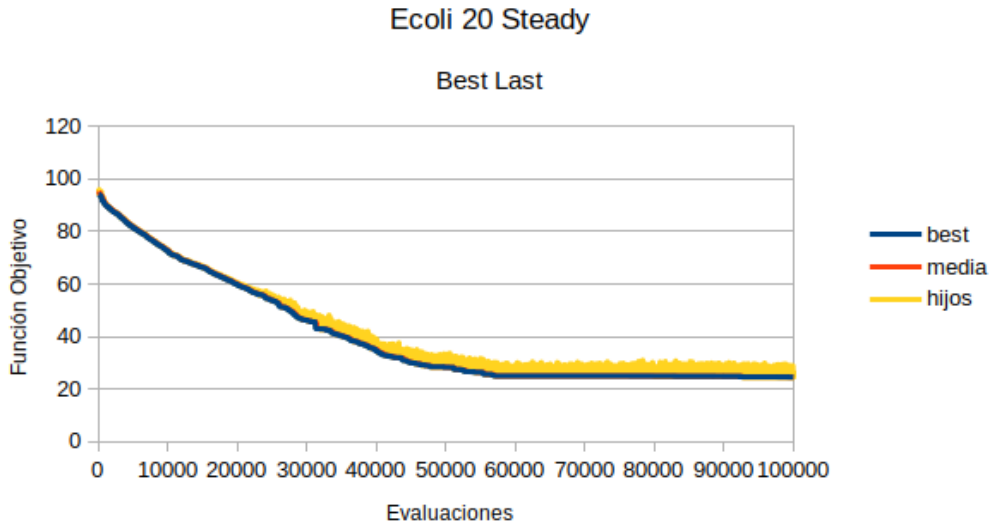
Cuadro 6: AGE (20 % de restricciones) OpCruce: BestLast

Todas las variaciones comparten resultados ciertamente similares. En términos temporales no se nota una diferencia significativa. Se puede apreciar cierta tendencia en el operador *BestFirst* de reducir *infeasibility* de forma más agresiva (menor media) sobre todo cuando hay menor número de restricciones. Se describan algunas comparaciones basadas en el *dataset Ecoli* por ser el más representativo en las diferencias. Para un 10 % de restricciones, el operador *BestLast* consigue la mejor solución. Para un 20 % de restricciones, el operador uniforme consigue la mejor solución. En general, si se fija en las medias el operador *BestFirst* suele tener los resultados más balanceados. Si no son los mejores son más cercanos en comparación al caso inverso. Como se supuso anteriormente, los algoritmos AGG van a converger de manera más lenta por lo que puede suponerse que operadores como *BestFirst* (que fuerzan una convergencia más rápida) pueden favorecerles de cierta manera. Sin embargo, los resultados han sido muy cercanos.

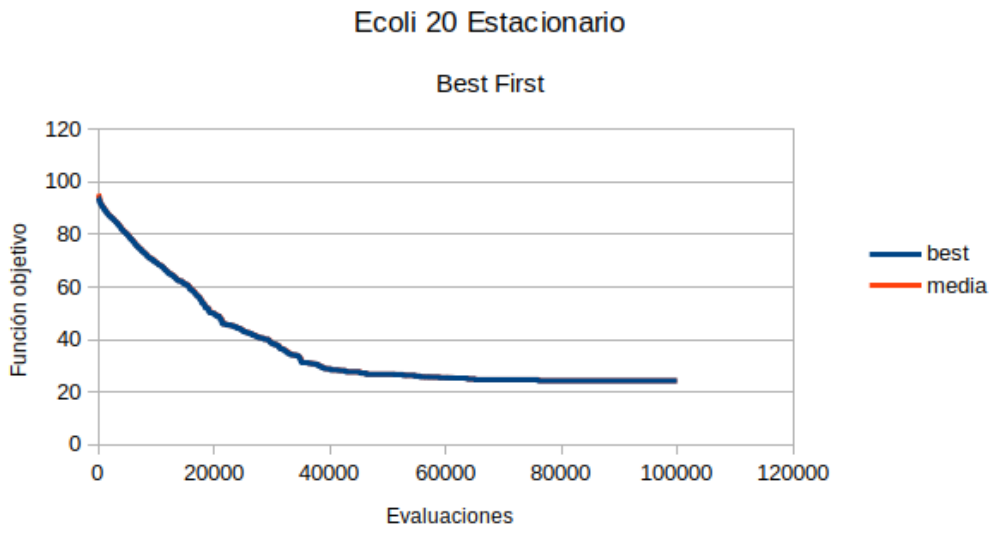
6.3.1.2. Convergencia en algoritmos genéticos estacionarios

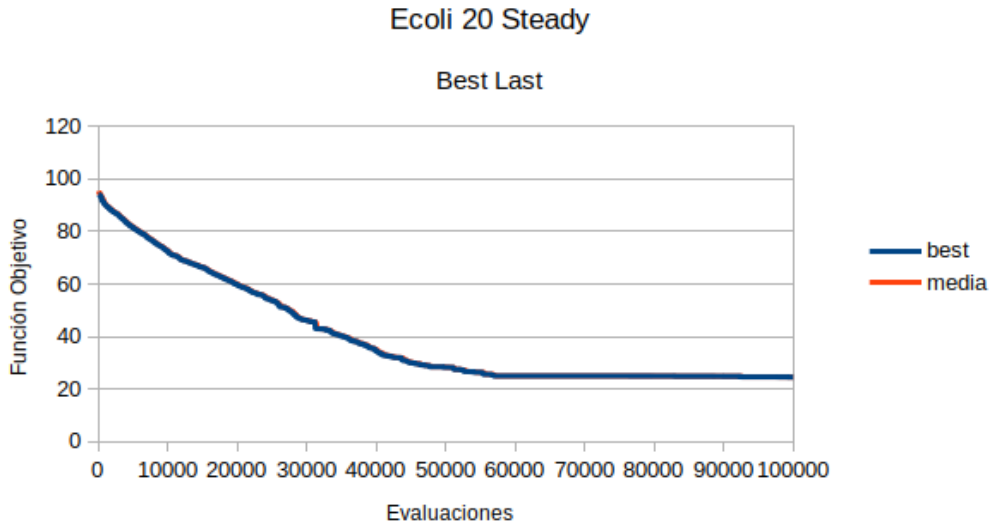
Se comienza viendo las gráficas de convergencia entre ambas variaciones del cruce por segmento fijo.



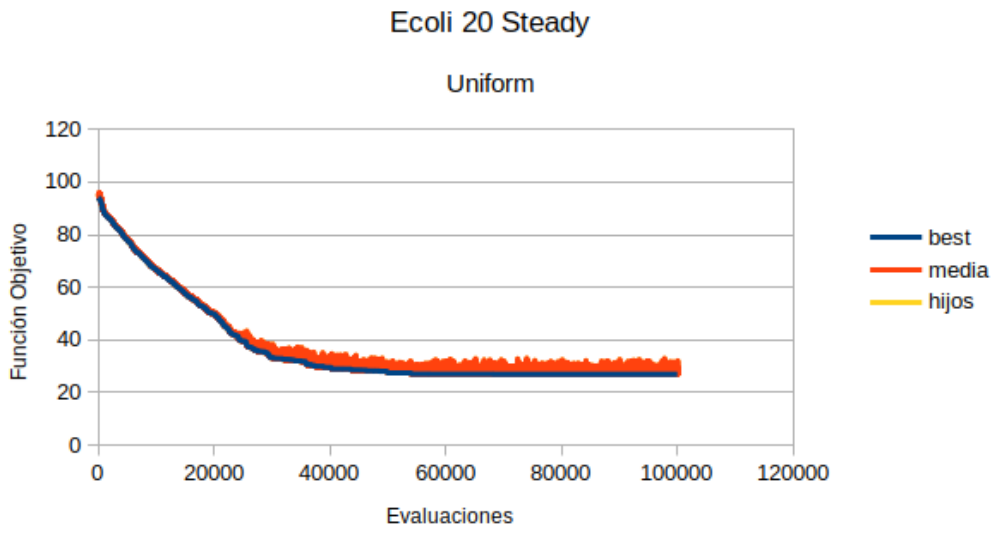


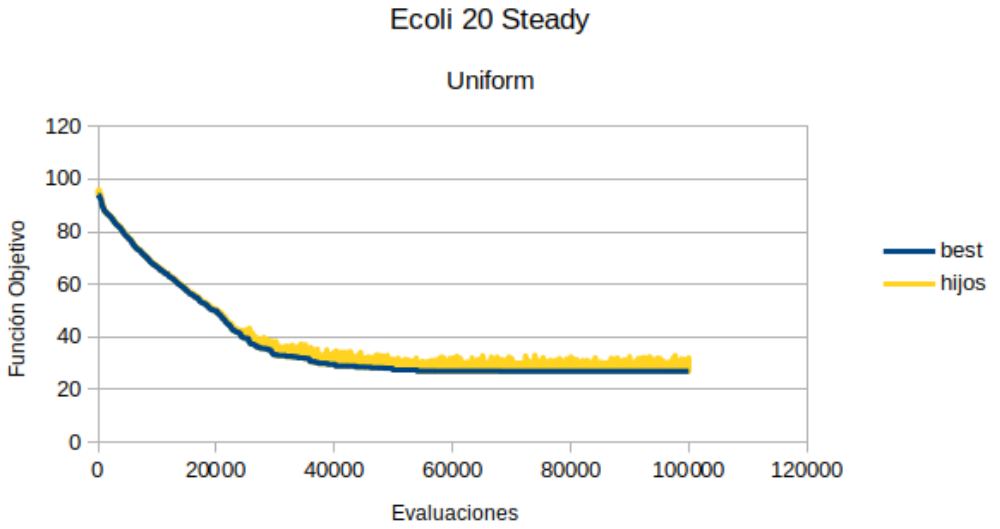
Como era de esperar se aprecia una diferencia en la convergencia entre ambas gráficas. El cruce *BestFirst* converge de manera más rápida. Se suprime la gráfica de hijos:





Se aprecia que la media de la función objetivo se mantiene cercana a la mejor solución. Aunque los hijos varíen, la población es bastante homogénea por lo que solo pasaran los hijos que también entre dentro de ese parecido. Se pasa a ver el último operador de cruce:





A diferencia de los anteriores, este operador provoca unos hijos muy variables y una media de la población muy variable también. Ya que los hijos se generan combinando de forma aleatoria los padres, el resultado puede favorecer exploración o explotación. Todo esto provoca una población poco homogénea que genera a su vez, hijos poco homogéneos. La gran variación podría también explicar el ratio de convergencia el cual se parece bastante al de *BestFirst*.

6.3.1.3. Algoritmos genéticos generacionales

Luego de ver los resultados al aplicar el esquema estacionario, se pasa al esquema

generacional. En este esquema se espera ver cambios más bruscos al sustituir la población entera. Se consigue algo de continuidad al aplicar elitismo y mantener la mejor solución. Puede que se beneficien más por operadores de cruce uniformes o incluso aquellos que favorezcan la exploración.

	Iris				Ecoli				Rand				NewThyroid			
	TasaC	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T
Seed 123	0.63	6.00	0.67	244.18	23.73	48.00	25.06	806.12	0.71	1.00	0.72	217.47	13.80	7.00	14.07	398.22
Seed 456	0.65	3.00	0.67	251.75	22.12	95.00	24.75	803.82	0.70	2.00	0.72	214.55	13.72	9.00	14.07	399.76
Seed 789	0.64	4.00	0.67	243.57	20.68	102.00	23.50	800.23	0.71	1.00	0.72	223.16	11.52	74.00	14.36	401.82
Seed 101112	0.67	0.00	0.67	246.98	22.27	89.00	24.73	800.32	0.70	2.00	0.72	221.41	13.68	10.00	14.07	402.32
Seed 131415	0.67	0.00	0.67	240.34	22.11	90.00	24.60	821.64	0.70	2.00	0.72	215.67	13.76	8.00	14.07	397.30
Media	0.65	2.60	0.67	245.36	22.18	84.80	24.53	806.43	0.70	1.60	0.72	218.45	13.29	21.60	14.12	399.88

Cuadro 7: AGG (10 % de restricciones) OpCruce: Uniforme

	Iris				Ecoli				Rand				NewThyroid			
	TasaC	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T
Seed 123	0.64	9.00	0.67	325.85	20.82	201.00	23.56	1201.53	0.71	2.00	0.72	291.43	14.17	6.00	14.29	557.11
Seed 456	0.63	11.00	0.67	328.47	21.31	192.00	23.93	1198.60	0.70	3.00	0.72	297.24	14.10	10.00	14.29	549.80
Seed 789	0.65	6.00	0.67	323.59	21.77	174.00	24.14	1187.85	0.70	3.00	0.72	290.71	14.03	14.00	14.29	556.90
Seed 101112	0.67	0.00	0.67	318.77	21.62	159.00	23.78	1192.08	0.70	4.00	0.72	294.16	14.04	13.00	14.29	549.01
Seed 131415	0.67	0.00	0.67	325.57	22.41	129.00	24.16	1180.44	0.70	5.00	0.72	294.16	14.14	8.00	14.29	553.32
Media	0.65	5.20	0.67	324.45	21.59	171.00	23.92	1192.10	0.70	3.40	0.72	293.54	14.10	10.20	14.29	553.23

Cuadro 8: AGG (20 % de restricciones) OpCruce: Uniforme

	Iris				Ecoli				Rand				NewThyroid			
	TasaC	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T
Seed 123	0.66	1.000	0.67	265.51	21.37	107.00	24.33	807.45	0.71	1.00	0.72	220.75	11.84	86.00	15.14	408.04
Seed 456	0.66	1.000	0.67	241.50	21.49	84.00	23.82	804.27	0.71	1.00	0.72	218.68	13.76	8.00	14.07	400.32
Seed 789	0.66	1.000	0.67	240.01	25.41	102.00	28.23	807.62	0.71	1.00	0.72	223.14	11.85	72.00	14.62	400.01
Seed 101112	0.66	1.000	0.67	236.68	22.70	73.00	24.71	814.12	0.71	1.00	0.72	219.65	11.78	76.00	14.69	401.48
Seed 131415	0.64	4.000	0.67	244.55	21.84	72.00	23.83	826.76	0.70	2.00	0.72	221.42	11.47	74.00	14.31	399.27
Media	0.66	1.60	0.67	245.65	22.56	87.60	24.98	812.05	0.71	1.20	0.72	220.73	12.14	63.20	14.56	401.82

Cuadro 9: AGG (10 % de restricciones) OpCruce: BestFirst

	Iris				Ecoli				Rand				NewThyroid			
	TasaC	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T
Seed 123	0.66	4.00	0.67	319.08	22.36	196.00	25.03	1188.16	0.70	4.00	0.72	302.70	14.10	10.00	14.29	563.48
Seed 456	0.66	2.00	0.67	315.57	21.86	184.00	24.36	1192.58	0.71	2.00	0.72	291.19	14.14	8.00	14.29	556.10
Seed 789	0.66	2.00	0.67	313.03	22.77	160.00	24.95	1191.96	0.71	2.00	0.72	298.72	14.21	4.00	14.29	565.32
Seed 101112	0.66	2.00	0.67	317.97	22.05	173.00	24.41	1205.73	0.71	2.00	0.72	300.15	14.12	9.00	14.29	554.59
Seed 131415	0.66	4.00	0.67	319.14	21.33	184.00	23.84	1211.17	0.70	4.00	0.72	297.90	14.12	9.00	14.29	564.83
Media	0.66	2.80	0.67	316.96	22.08	179.40	24.52	1197.92	0.71	2.80	0.72	298.13	14.14	8.00	14.29	560.86

Cuadro 10: AGG (20 % de restricciones) OpCruce: BestFirst

	Iris				Ecoli				Rand				NewThyroid			
	TasaC	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T
Seed 123	0.66	2.00	0.67	241.38	25.45	156.00	29.76	825.53	0.70	2.00	0.72	217.35	13.72	9.00	14.07	399.75
Seed 456	0.66	1.00	0.67	238.49	26.34	147.00	30.40	832.41	0.71	1.00	0.72	218.50	13.76	8.00	14.07	397.27
Seed 789	0.66	1.00	0.67	239.29	31.37	348.00	40.99	815.04	0.71	1.00	0.72	232.55	13.68	10.00	14.07	406.35
Seed 101112	0.66	1.00	0.67	241.65	25.95	166.00	30.54	805.57	0.70	2.00	0.72	214.75	13.80	7.00	14.07	402.61
Seed 131415	0.65	3.00	0.67	240.82	31.49	320.00	40.34	848.87	0.70	2.00	0.72	226.89	13.72	9.00	14.07	398.30
Media	0.66	1.60	0.67	240.33	28.12	227.40	34.41	825.48	0.70	1.60	0.72	222.01	13.74	8.60	14.07	400.85

Cuadro 11: AGG (10 % de restricciones) OpCruce: BestLast

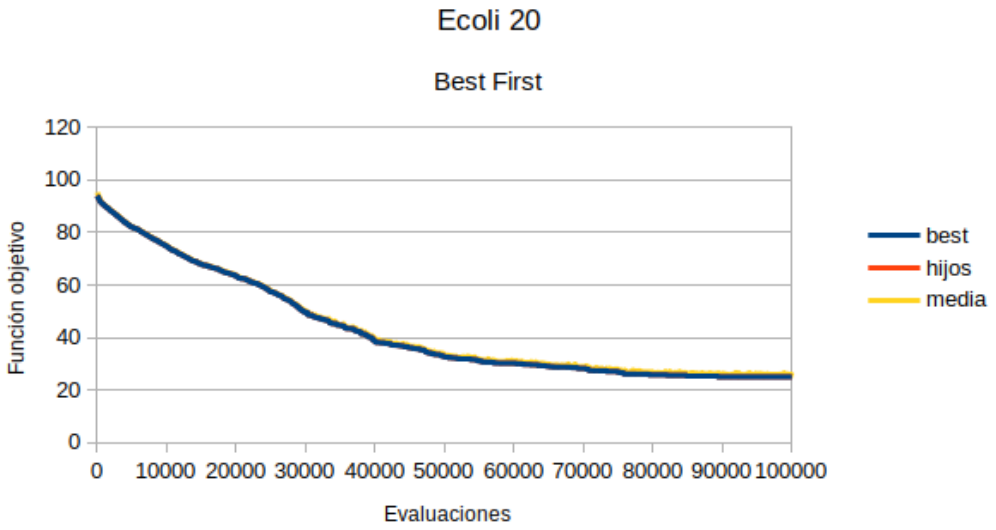
	Iris				Ecoli				Rand				NewThyroid			
	TasaC	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T
Seed 123	0.66	2.00	0.67	233.63	24.16	224.00	27.21	1225.72	0.70	4.00	0.72	294.72	11.72	177.00	15.03	571.92
Seed 456	0.66	2.00	0.67	248.94	23.21	214.00	26.12	1213.90	0.71	2.00	0.72	293.67	11.89	175.00	15.17	572.77
Seed 789	0.66	2.00	0.67	243.94	22.74	196.00	25.40	1192.94	0.71	2.00	0.72	294.57	14.12	9.00	14.29	557.51
Seed 101112	0.66	2.00	0.67	238.31	22.31	175.00	24.69	1202.59	0.70	3.00	0.72	304.56	14.14	8.00	14.29	572.53
Seed 131415	0.66	2.00	0.67	231.28	25.43	214.00	28.34	1223.05	0.70	5.00	0.72	305.84	11.86	178.00	15.19	561.72
Media	0.66	2.00	0.67	239.22	23.57	204.60	26.35	1211.64	0.70	3.20	0.72	298.67	12.75	109.40	14.79	567.29

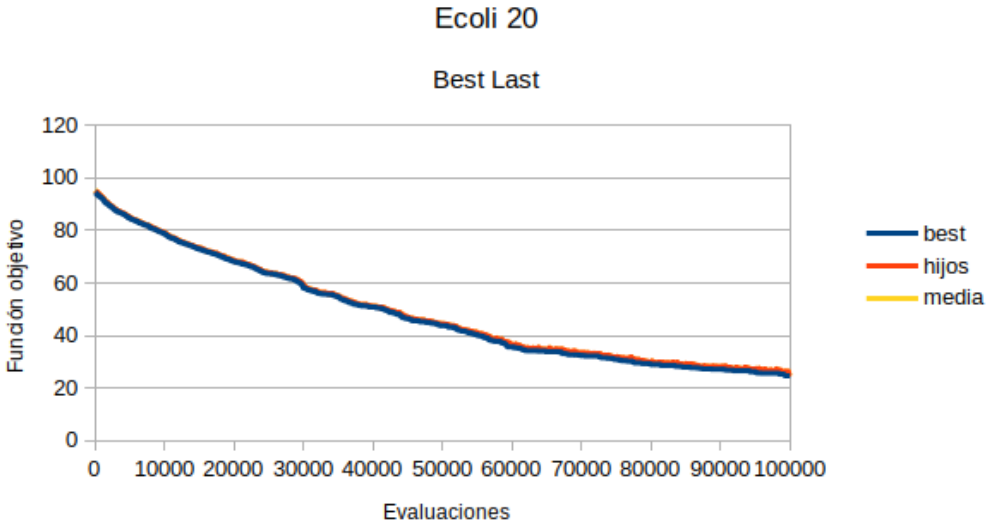
Cuadro 12: AGG (20 % de restricciones) OpCruce: BestLast

Los resultados son bastante ilustradores. Contrario a la suposición anterior, el operador *BestLast* provoca los peores resultados de todos. Una posibilidad podría ser que al crear todas las nuevas soluciones favoreciendo la exploración, esta converja de una forma muy lenta. Esto significaría que se podría llegar a mejores soluciones si se hubiesen realizado un mayor número de iteraciones. Esto podría explicar porqué si que se han llegado a las mismas soluciones en los otros conjuntos de datos (más pequeños). En esta ocasión, se obtiene una clara ventaja en el uso del operador uniforme. Este operador no solo produce las mejores soluciones, sino también la mejor media de estas.

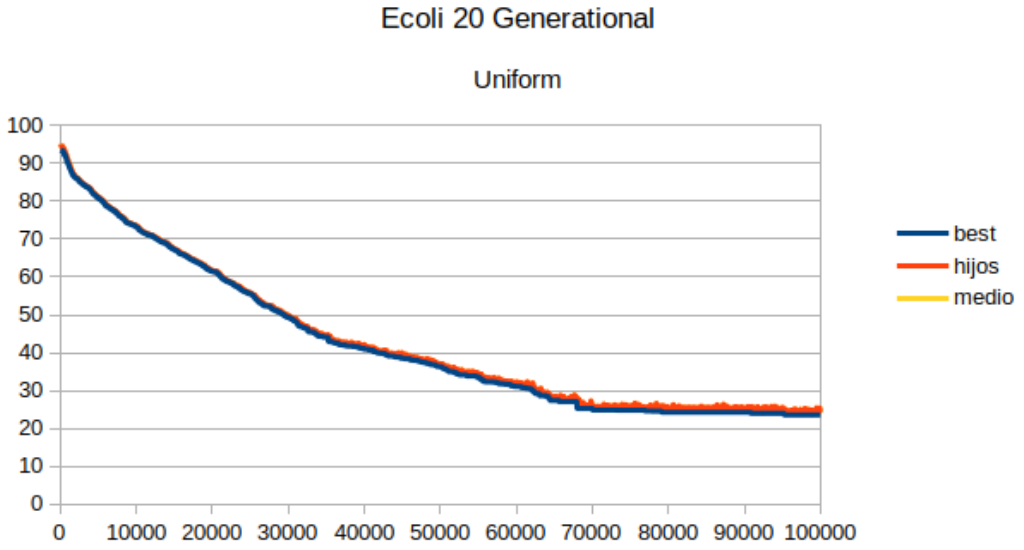
6.3.1.4. Convergencia en algoritmos genéticos generacionales

De nuevo, se comenzará viendo la convergencia entre ambas variaciones del cruce por segmento fijo:





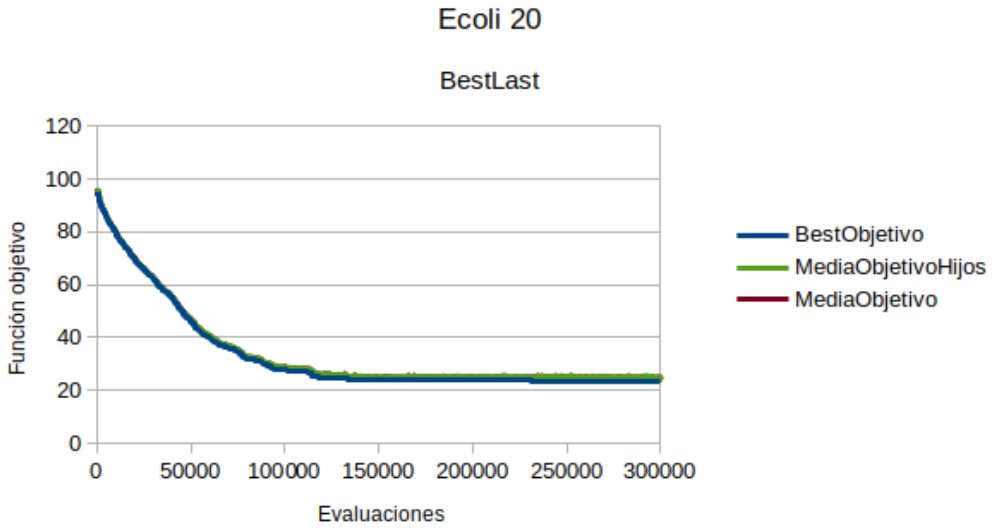
Estas gráficas muestran como el esquema generacional converge de forma más lenta con respecto al esquema estacionario. No solo eso, los hijos no varían tanto con respecto a la mejor solución de la población. Por último, en la gráfica de *BestLast* se puede confirmar la sospecha de que este algoritmo no había acabado de converger y por ello no daba las mejores soluciones. Se pasa entonces a la gráfica del cruce uniforme:



Sorprendentemente, esta gráfica parece converger más rápidamente incluso que *BestFirst*. Puede pasar por varios motivos incluyendo aleatoriedad. Sin embargo, la combinación de exploración y explotación puede dar lugar a hijos que den mayor posibilidad a la creación de nuevos hijos (no posibles aún en *BestFirst*) que convergen de manera mas rápida.

6.3.1.5. Aumento de evaluaciones

El operador *BestLast* se ha supuesto que el número de evaluaciones no era suficiente. Se quiere mostrar los efectos de un gran número de evaluaciones puede tener en la población. Se puede predecir que se obtendrá una mejor solución debido a la anterior gráfica. Una gráfica también mostrará el límite de evaluaciones útiles en este caso.



Como se puede apreciar, se ha triplicado el número de evaluaciones. Comienza a llegar al óptimo local sobre las 130.000 evaluaciones. También se aprecia que se produce una pequeña mejora en la mejor solución tras 230.000 evaluaciones. Esto nos muestra los límites de evaluaciones para este caso. Es muy poco probable que se siga reduciendo luego de estas evaluaciones. Luego no todo depende del número de evaluaciones, también depende de la forma en que converja y si pasa por las soluciones necesarias.

6.3.2. Análisis del algoritmo meméticos

Se describen a continuación los datos obtenido a partir de los algoritmos meméticos. El algoritmo memético ha seguido el esquema de evolución estacionario con elitismo. Este algoritmo tiene 3 variaciones (explicadas anteriormente) según la selección que se realice para la aplicación de la búsqueda local. Aunque el operador de cruce que ha traído mejores resultados haya sido *BestFirst*, se ha decidido volver a probar con los 3 operadores. La mayor exploración de *BestLast* o la distribución de *Uniform* pueden traer posibles beneficies a la solución final.

6.3.2.1. Algoritmo memético *All*

Se comienza describiendo los resultados de la variación más sencilla. Esta variación selecciona todos los cromosomas de la población actual. Se puede esperar la mayor convergencia en esta variación, sobre todo al comienzo.

	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	71.88	22.24	151.00	26.42	37.93	0.72	0.00	0.72	222.37	10.90	97.00	14.62	90.87
Seed 456	0.67	0.00	0.67	72.21	22.29	48.00	23.62	38.62	0.72	0.00	0.72	219.05	13.87	6.00	14.10	91.02
Seed 789	0.67	0.00	0.67	73.23	21.09	119.00	24.38	37.60	0.72	0.00	0.72	221.74	13.62	2.00	13.69	90.13
Seed 101112	0.67	0.00	0.67	73.16	23.35	79.00	25.53	38.48	0.72	0.00	0.72	219.37	13.62	0.00	13.62	94.67
Seed 131415	0.67	0.00	0.67	72.59	21.50	97.00	24.18	38.91	0.72	0.00	0.72	216.67	10.81	98.00	14.56	91.27
Media	0.67	0.00	0.67	72.61	22.09	98.80	24.82	38.31	0.72	0.00	0.72	219.84	12.56	40.60	14.12	91.59

Cuadro 13: AM All (10 % de restricciones) OpCruce: Uniform

	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	106.51	19.66	265.00	23.27	62.28	0.72	0.00	0.72	322.15	14.24	5.00	14.33	140.25
Seed 456	0.67	0.00	0.67	103.66	19.54	293.00	23.53	60.06	0.72	0.00	0.72	320.21	10.82	256.00	15.61	138.85
Seed 789	0.67	0.00	0.67	101.98	20.66	248.00	24.04	60.63	0.72	0.00	0.72	318.70	14.24	5.00	14.33	134.87
Seed 101112	0.67	0.00	0.67	101.98	22.22	226.00	25.30	61.88	0.72	0.00	0.72	323.46	10.75	223.00	14.93	135.78
Seed 131415	0.67	0.00	0.67	102.98	21.63	182.00	24.11	59.24	0.72	0.00	0.72	329.90	10.88	241.00	15.39	140.39
Media	0.67	0.00	0.67	103.42	20.75	242.80	24.05	60.82	0.72	0.00	0.72	322.88	12.18	146.00	14.92	138.03

Cuadro 14: AM All (20 % de restricciones) OpCruce: Uniform

	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	72.46	26.04	118.00	29.31	39.98	0.72	0.00	0.72	69.21	10.81	98.00	14.57	92.94
Seed 456	0.67	0.00	0.67	72.56	24.29	177.00	29.19	37.95	0.72	0.00	0.72	68.91	10.79	95.00	14.43	91.21
Seed 789	0.67	0.00	0.67	73.87	21.15	192.00	26.46	37.99	0.72	0.00	0.72	68.92	13.48	11.00	13.91	91.65
Seed 101112	0.67	0.00	0.67	72.78	20.31	226.00	26.56	36.69	0.72	0.00	0.72	70.34	13.83	234.0,	13.90	91.17
Seed 131415	0.67	0.00	0.67	72.00	21.06	105.00	23.96	38.55	0.72	0.00	0.72	68.69	10.87	98.00	14.63	95.47
Media	0.67	0.00	0.67	72.73	22.57	163.60	27.09	38.23	0.72	0.00	0.72	69.21	11.96	75.50	14.29	92.49

Cuadro 15: AM All (10 % de restricciones) OpCruce: BestFirst

	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	102.07	21.97	172.00	24.31	62.17	0.72	0.00	0.72	99.63	14.29	0.00	14.29	141.09
Seed 456	0.67	0.00	0.67	105.08	21.41	231.00	24.56	63.54	0.72	0.00	0.72	98.56	14.29	0.00	14.29	135.91
Seed 789	0.67	0.00	0.67	104.78	22.24	123.00	23.91	61.45	0.72	0.00	0.72	100.39	10.83	234.0,	15.21	139.74
Seed 101112	0.67	0.00	0.67	103.80	20.36	230.00	23.50	60.54	0.71	0.00	0.71	102.05	14.17	0.00	14.17	138.93
Seed 131415	0.67	0.00	0.67	105.27	21.58	401.00	27.04	61.42	0.72	0.00	0.72	100.99	10.87	231.0,	15.20	136.23
Media	0.67	0.00	0.67	104.20	21.51	231.40	24.66	61.82	0.72	0.00	0.72	100.32	12.89	0.00	14.63	138.38

Cuadro 16: AM All (20 % de restricciones) OpCruce: BestFirst

	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	74.22	24.33	179.00	29.28	38.21	0.71	0.00	0.71	69.17	13.90	0.00	13.90	91.58
Seed 456	0.67	0.00	0.67	72.64	21.31	75.00	23.39	38.98	0.72	0.00	0.72	68.67	13.60	2.00	13.67	89.56
Seed 789	0.67	0.00	0.67	74.46	23.87	113.00	27.00	38.32	0.72	0.00	0.72	69.44	10.89	95.00	14.54	93.25
Seed 101112	0.67	0.00	0.67	72.55	21.84	91.00	24.35	39.90	0.72	0.00	0.72	69.04	10.77	100.00	14.60	91.13
Seed 131415	0.67	0.00	0.67	72.61	21.37	116.00	24.58	37.86	0.72	0.00	0.72	68.98	10.87	96.00	14.56	91.64
Media	0.67	0.00	0.67	73.30	22.54	114.80	25.72	38.65	0.71	0.00	0.71	69.06	12.01	58.60	14.25	91.43

Cuadro 17: AM All (10 % de restricciones) OpCruce: BestLast

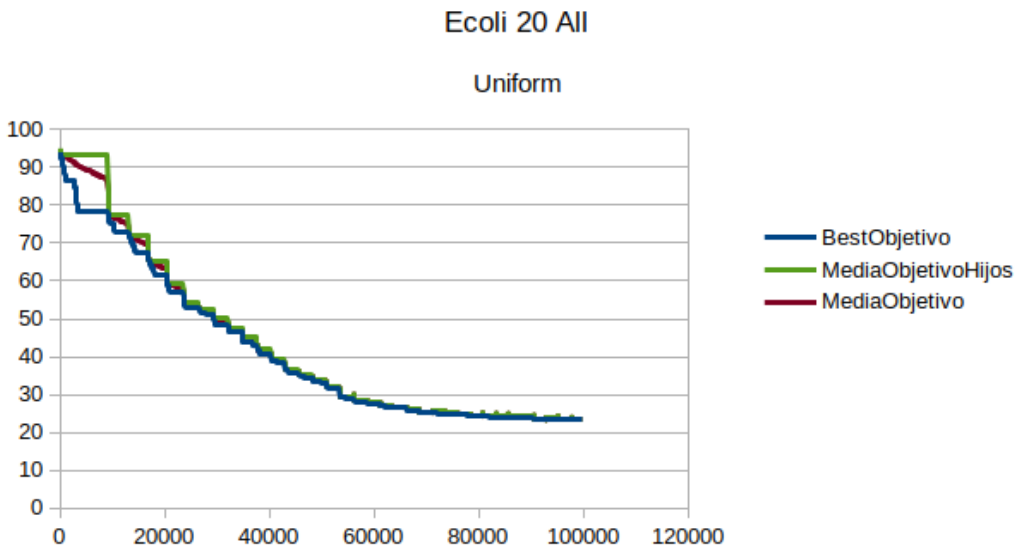
	Iris				Ecoli				Rand				NewThyroid			
	TasaC	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**	***TasaC***	***Inf***	***Agr***	**T**
Seed 123	0.67	0.00	0.67	107.09	22.50	111.00	24.01	63.20	0.72	0.00	0.72	98.63	10.87	235.00	15.27	136.34
Seed 456	0.67	0.00	0.67	102.77	22.73	203.00	25.50	59.30	0.71	0.00	0.71	100.87	13.99	14.00	14.25	134.83
Seed 789	0.67	0.00	0.67	103.14	22.37	205.00	25.16	59.93	0.72	0.00	0.72	100.41	10.83	234.00	15.21	140.35
Seed 101112	0.67	0.00	0.67	104.90	21.36	495.00	28.10	58.99	0.71	0.00	0.71	99.49	10.80	260.00	15.66	135.61
Seed 131415	0.67	0.00	0.67	105.53	21.65	144.00	23.61	61.44	0.72	0.00	0.72	106.28	10.83	234.00	15.21	139.77
Media	0.67	0.00	0.67	104.69	22.12	231.60	25.28	60.57	0.71	0.00	0.71	101.14	11.46	195.40	15.12	137.38

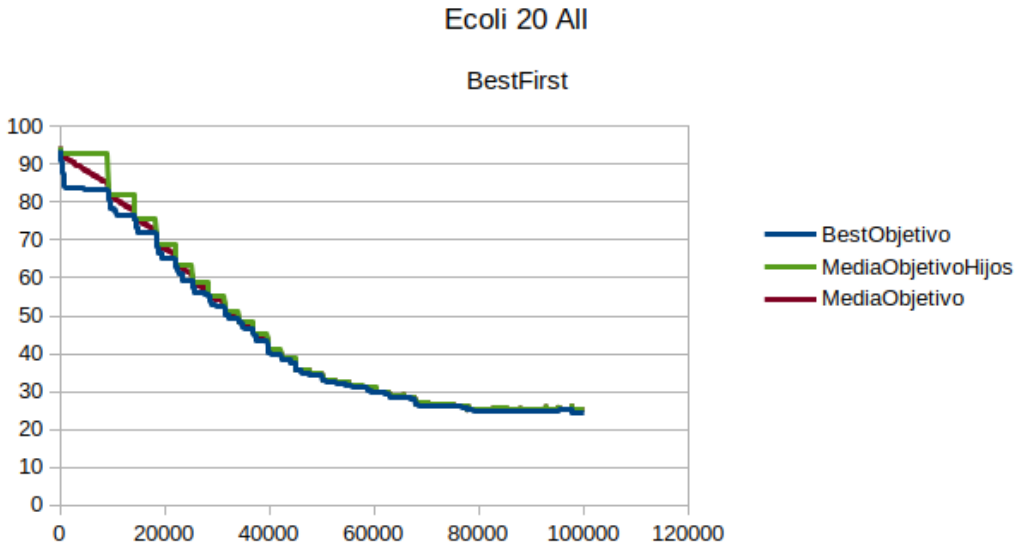
Cuadro 18: AM All (20 % de restricciones) OpCruce: BestLast

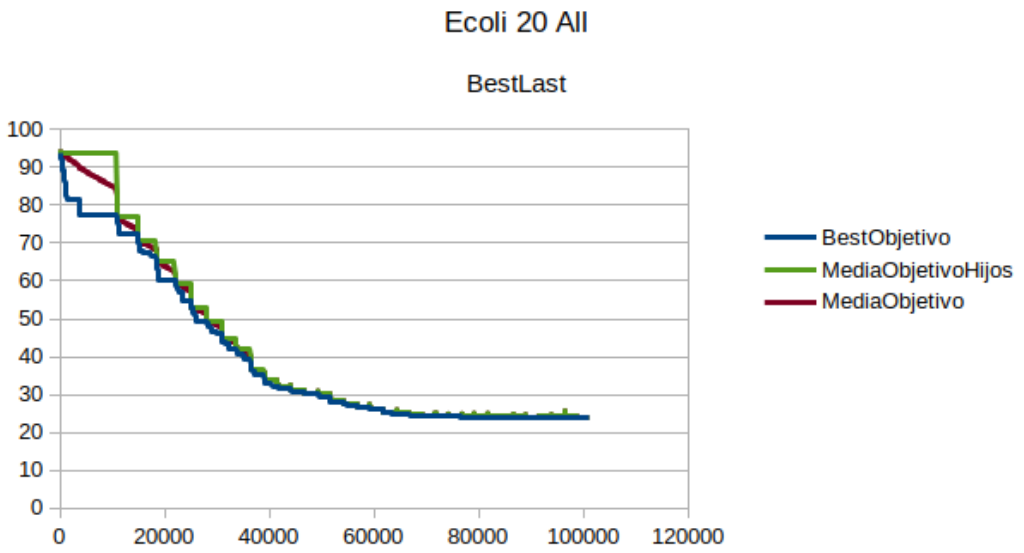
En esta caso las mejores medias se consiguen a través del operador uniforme. El operador *BestFirst* no consigue resultados muy óptimos con 10 % de restricciones. Esto se puede deber a la aleatoriedad de las semillas o a la mayor convergencia. Como ya se ha hablado, esta variación puede provocar bastante convergencia en la búsqueda local. Al añadir un operado también centrado en converger, los resultados pueden ser óptimos locales. Esto podría explicar por qué los resultados de este operador de cruce son mejores con mayor nivel de restricción. Esta actúa como una guía acercándolo a mejores opciones evitando óptimos locales.

6.3.2.2. Convergencia en algoritmo memético *All*

Se exponen las diferentes gráficas de convergencia:

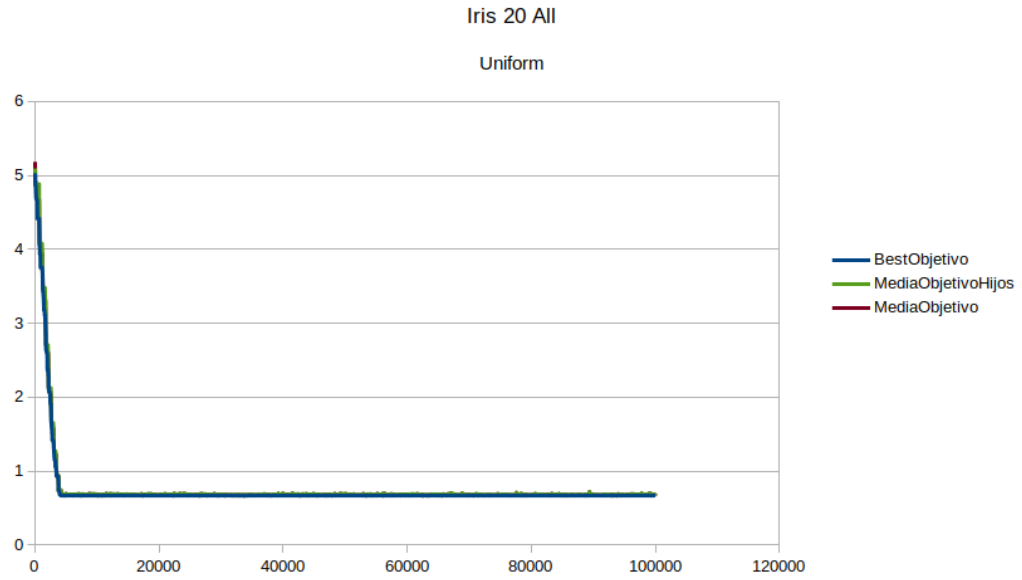

Ecoli 20 All
Uniform
100
90
80
70
60
50
40
30
20
10
0
0
20000
40000
60000
80000
100000
120000
BestObjetivo
MediaObjetivoHijos
MediaObjetivo


Ecoli 20 All
BestFirst
100
90
80
70
60
50
40
30
20
10
0
0
20000
40000
60000
80000
100000
120000
BestObjetivo
MediaObjetivoHijos
MediaObjetivo



Estás gráficas dan lugar a un análisis muy interesante. Se aprecia como el operador *BestLast* parece converger más rápido que *BestFirst*. Esto contradice las suposiciones de los operadores. Sin embargo, al ver las gráficas se puede ver el tamaño de las 'cajas' o 'escalones' formados. Estos escalones se forman por una mejora rápida, es decir, la búsqueda local. Al fijarnos en el tamaño de los escalones también se puede ver como *BestLast* supera a *BestFirst*. Estos escalones representan cuánto ha podido mejorar la búsqueda local la población elegida. Al usar operadores que favorecen la exploración, permitimos crear una población diversa. Esto significa que es fácilmente optimizable, por tanto el número de óptimos locales en los genes de los cromosomas será mucho menor. En cambio, *BestFirst*, favorece la explotación. La población creada tendrá más calidad y en la búsqueda local, el contador de errores subirá rápidamente.

En las tablas también se puede apreciar otra anormalidad. Los conjuntos de datos pequeños tardan lo mismo e incluso más que los conjuntos de datos grandes. Esto se puede explicar fácilmente con la gráfica de convergencia de estos *datasets*.



Como se puede ver en la gráfica, la función converge rápidamente alcanzando un óptimo local. Tras esto, el algoritmo genético seguirá creando nuevas generaciones. El proceso de crear generaciones consume mas tiempo por evaluación realizada. Los *datasets* como *Ecoli*, no convergen tan rápidamente con una búsqueda local. Dado que la búsqueda local se guía por el número de fallos para parar, esto significa que podrá realizar muchos mas cambios en conjuntos de datos que no estén en un óptimo local. Esto resulta en que los conjuntos grandes gastan una mayor cantidad de evaluaciones en la búsqueda local, acelerando entonces el tiempo total consumido.

6.3.2.3. Algoritmo memético *Random*

La siguiente variación selecciona un subconjunto de la población para aplicar la búsqueda local. En este caso la probabilidad es $P_S = 0.1$, con una población de 10 cromosomas. Se puede esperar una mayor diversidad en los resultados así como menor grado de convergencia.

	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	227.98	19.73	95.00	22.35	287.82	0.72	0.00	0.72	222.37	10.82	112.00	15.12	361.00
Seed 456	0.67	0.00	0.67	232.85	19.64	134.00	23.35	293.07	0.72	0.00	0.72	219.05	10.89	101.00	14.77	363.29
Seed 789	0.67	0.00	0.67	230.41	19.68	107.00	22.64	292.41	0.72	0.00	0.72	221.74	10.87	97.00	14.59	357.84
Seed 101112	0.67	0.00	0.67	227.66	22.26	58.00	23.86	290.62	0.72	0.00	0.72	219.37	10.90	97.00	14.62	354.09
Seed 131415	0.67	0.00	0.67	235.74	19.63	92.00	22.18	295.13	0.72	0.00	0.72	216.67	10.89	98.00	14.65	357.60
Media	0.67	0.00	0.67	230.93	20.19	97.20	22.88	291.81	0.72	0.00	0.72	219.84	10.87	101.00	14.75	358.76

Cuadro 19: AM Random (10 % de restricciones) OpCruce: Uniform

	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	330.81	21.83	186.00	24.36	479.73	0.72	0.00	0.72	322.15	14.29	0.00	14.29	533.99
Seed 456	0.67	0.00	0.67	332.06	21.58	167.00	23.85	477.51	0.72	0.00	0.72	320.21	10.81	231.00	15.14	546.54
Seed 789	0.67	0.00	0.67	331.84	19.67	252.00	23.10	464.80	0.72	0.00	0.72	318.70	14.29	0.00	14.29	546.01
Seed 101112	0.67	0.00	0.67	329.19	22.03	157.00	24.17	489.64	0.72	0.00	0.72	323.46	10.82	256.00	15.61	541.27
Seed 131415	0.67	0.00	0.67	342.99	19.63	222.00	22.65	469.07	0.72	0.00	0.72	329.90	10.89	233.00	15.26	571.28
Media	0.67	0.00	0.67	333.38	20.95	196.80	23.63	476.15	0.72	0.00	0.72	322.88	12.22	144.00	14.92	547.82

Cuadro 20: AM Random (20 % de restricciones) OpCruce: Uniform

	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	232.72	21.28	107.00	24.24	294.31	0.72	0.00	0.72	219.77	13.83	6.00	14.07	360.41
Seed 456	0.67	0.00	0.67	229.28	22.30	87.00	24.70	295.93	0.72	0.00	0.72	222.11	13.83	6.00	14.07	360.84
Seed 789	0.67	0.00	0.67	227.32	22.05	78.00	24.21	286.06	0.72	0.00	0.72	217.78	13.83	6.00	14.07	351.88
Seed 101112	0.67	0.00	0.67	233.00	22.01	62.00	23.73	296.17	0.72	0.00	0.72	222.65	13.83	6.00	14.07	350.40
Seed 131415	0.67	0.00	0.67	229.41	19.54	125.00	22.99	293.51	0.72	0.00	0.72	218.08	10.90	97.00	14.62	362.47
Media	0.67	0.00	0.67	230.35	21.44	91.80	23.97	293.20	0.72	0.00	0.72	220.08	13.25	24.20	14.18	357.20

Cuadro 21: AM Random (10 % de restricciones) OpCruce: BestFirst

	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	341.49	19.43	239.00	22.68	484.60	0.72	0.00	0.72	338.66	14.29	0.00	14.29	548.15
Seed 456	0.67	0.00	0.67	336.86	21.63	238.00	24.88	474.02	0.72	0.00	0.72	329.04	14.29	0.00	14.29	533.24
Seed 789	0.67	0.00	0.67	334.48	19.64	277.00	23.41	476.26	0.72	0.00	0.72	319.98	14.29	0.00	14.29	557.24
Seed 101112	0.67	0.00	0.67	332.94	21.87	156.00	23.99	469.52	0.72	0.00	0.72	333.83	14.29	0.00	14.29	566.83
Seed 131415	0.67	0.00	0.67	336.94	19.67	296.00	23.70	470.15	0.72	0.00	0.72	323.14	14.29	0.00	14.29	556.25
Media	0.67	0.00	0.67	336.54	20.45	241.20	23.73	474.91	0.72	0.00	0.72	328.93	14.29	0.00	14.29	552.34

Cuadro 22: AM Random (20 % de restricciones) OpCruce: BestFirst

	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	234.57	21.93	67.00	23.78	288.62	0.72	0.00	0.72	223.90	13.83	6.00	14.07	352.46
Seed 456	0.67	0.00	0.67	233.40	19.76	105.00	22.66	294.01	0.72	0.00	0.72	221.34	10.89	95.00	14.54	354.66
Seed 789	0.67	0.00	0.67	239.00	21.52	108.00	24.51	289.23	0.72	0.00	0.72	218.07	13.83	6.00	14.07	350.19
Seed 101112	0.67	0.00	0.67	232.80	19.70	122.00	23.07	302.14	0.72	0.00	0.72	221.58	13.83	6.00	14.07	355.98
Seed 131415	0.67	0.00	0.67	231.27	22.27	66.00	24.10	302.93	0.72	0.00	0.72	217.81	10.90	99.00	14.69	366.11
Media	0.67	0.00	0.67	234.21	21.04	93.60	23.62	295.39	0.72	0.00	0.72	220.54	12.66	42.40	14.29	355.88

Cuadro 23: AM Random (10 % de restricciones) OpCruce: BestLast

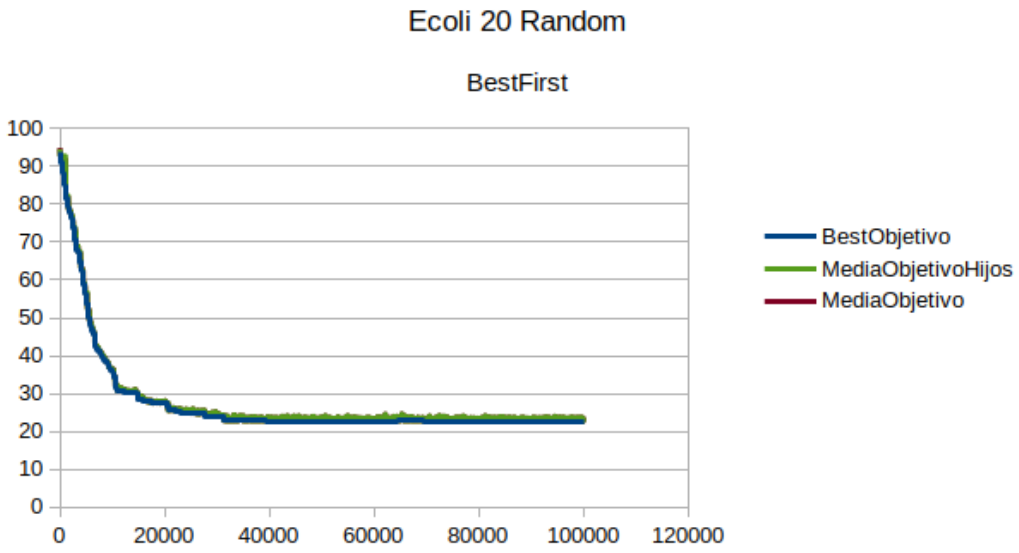
	Iris				Ecoli				Rand				NewThyroid			
	TasaC	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T
Seed 123	0.67	0.00	0.67	331.93	19.57	246.00	22.92	473.61	0.72	0.00	0.72	320.51	10.87	231.00	15.19	544.82
Seed 456	0.67	0.00	0.67	332.12	19.67	246.00	23.02	470.85	0.72	0.00	0.72	320.69	10.80	225.00	15.01	551.13
Seed 789	0.67	0.00	0.67	336.07	21.82	151.00	23.88	476.29	0.72	0.00	0.72	329.39	14.29	0.00	14.29	537.64
Seed 101112	0.67	0.00	0.67	335.92	21.47	201.00	24.21	485.38	0.72	0.00	0.72	317.09	14.29	0.00	14.29	557.15
Seed 131415	0.67	0.00	0.67	334.87	19.61	266.00	23.23	468.78	0.72	0.00	0.72	323.94	14.29	0.00	14.29	544.41
Media	0.67	0.00	0.67	334.18	20.43	222.00	23.45	474.98	0.72	0.00	0.72	322.32	12.91	91.20	14.61	547.03

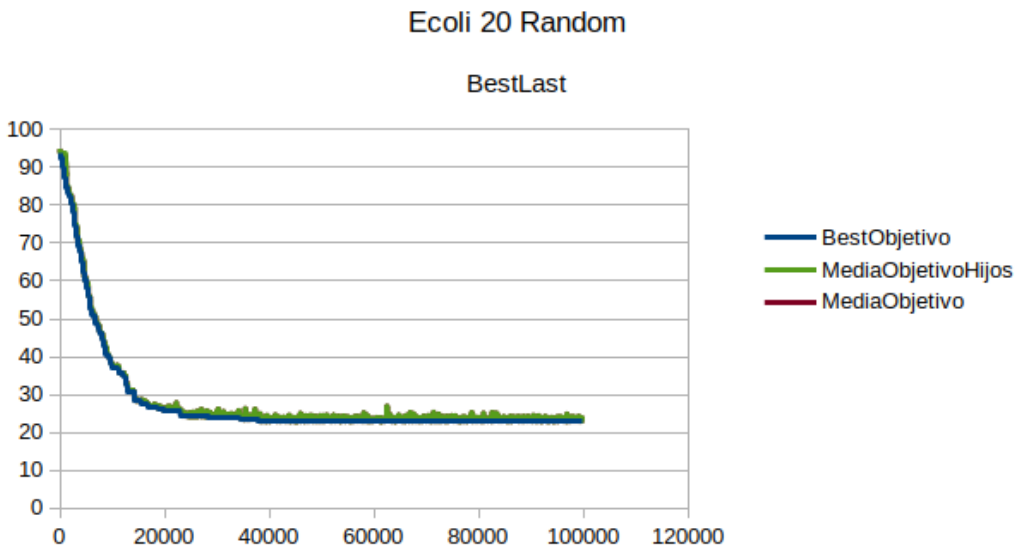
Cuadro 24: AM Random (20 % de restricciones) OpCruce: BestLast

En esta ocasión se repite ligeramente el último patrón. El operador de cruce uniforme proporciona las mejores soluciones de media para 'Ecoli'. El operador *BestFirst* consigue soluciones muy cercanas a las mejores cuanto mayor sean las restricciones. Por otro lado el operador *BestFirst* proporciona mejores soluciones para 'NewThyroid'. Se podría sospechar que el operador *BestFirst* puede ser mas adecuado para *datasets* no muy grandes. Esto pondría ser explicado si se tiene en cuenta que el número de óptimos locales en el espacio de soluciones aumenta de forma exponencial. Un espacio de soluciones menor podría evitar más óptimos locales y por tanto llegar a la solución óptima (o una de mayor calidad) más fácilmente.

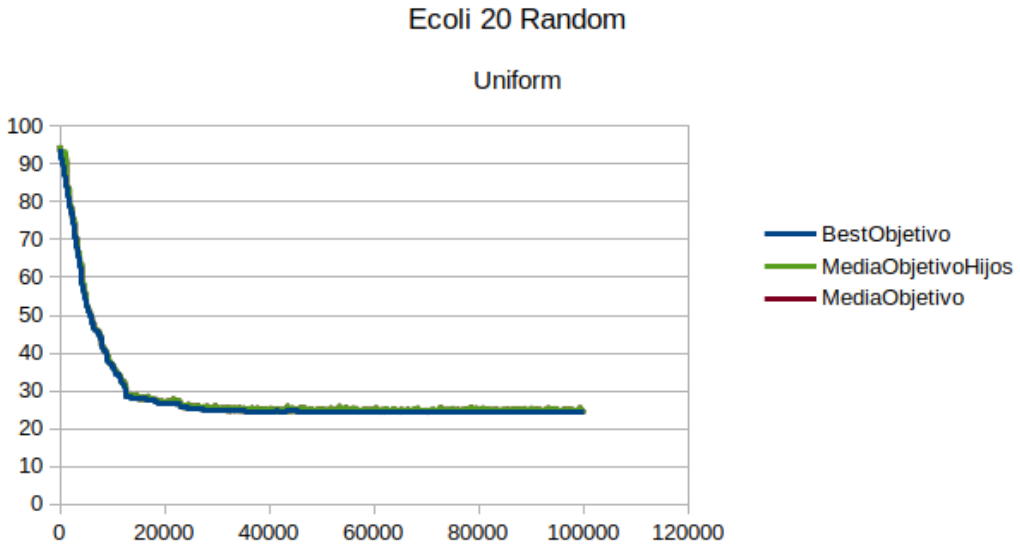
6.3.2.4. Convergencia en algoritmo memético *Random*

De nuevo, se comenzará viendo la convergencia entre ambas variaciones del cruce por segmento fijo:





Y por último el cruce uniforme:



A diferencia del anterior ejemplo. El grado de convergencia de *BestFirst* es mayor que *BestLast*. A pesar de que se explore más entre las búsquedas locales, esta solo afecta a una pequeña parte de la población. No tendrán un efecto tan drástico en la media total de la población.

6.3.2.5. Algoritmo memético *Best*

La siguiente variación selecciona un subconjunto de la población para aplicar la búsqueda local. Este subconjunto estará formado por los $N * P$ mejores individuos. P se refiere al porcentaje del tamaño de la población. En esto caso $P = 0.1$. En este caso, ya que se esta favoreciendo explotación con el método de selección de la búsqueda local, un operador que favorezca la exploración podría ser la mejor opción. Se muestran los datos:

	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	230.21	19.66	125.00	23.12	292.10	0.72	0.00	0.72	217.29	10.81	110.00	15.03	358.51
Seed 456	0.67	0.00	0.67	233.92	19.57	115.00	22.75	296.07	0.72	0.00	0.72	214.09	13.83	6.00	14.07	356.32
Seed 789	0.67	0.00	0.67	229.32	19.64	120.00	22.96	284.73	0.72	0.00	0.72	215.79	10.89	98.00	14.65	358.16
Seed 101112	0.67	0.00	0.67	235.22	19.68	99.00	22.42	291.43	0.72	0.00	0.72	216.95	10.88	97.00	14.60	363.08
Seed 131415	0.67	0.00	0.67	233.96	19.70	116.00	22.91	293.21	0.72	0.00	0.72	212.99	10.82	108.00	14.96	350.05
Media	0.67	0.00	0.67	232.52	19.65	115.00	22.83	291.51	0.72	0.00	0.72	215.42	11.45	83.80	14.66	357.22

Cuadro 25: AM Best (10 % de restricciones) OpCruce: Uniform

	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	333.51	19.53	274.00	23.26	471.03	0.72	0.00	0.72	331.50	14.29	0.00	14.29	537.11
Seed 456	0.67	0.00	0.67	330.67	19.62	278.00	23.41	477.45	0.72	0.00	0.72	324.84	10.82	264.00	15.76	551.12
Seed 789	0.67	0.00	0.67	336.65	21.81	152.00	23.88	466.24	0.72	0.00	0.72	323.08	10.90	239.00	15.37	571.20
Seed 101112	0.67	0.00	0.67	334.73	21.50	209.00	24.35	473.58	0.72	0.00	0.72	321.35	10.82	258.00	15.65	544.41
Seed 131415	0.67	0.00	0.67	329.73	19.56	278.00	23.35	469.58	0.72	0.00	0.72	323.31	10.81	255.00	15.59	556.73
Media	0.67	0.00	0.67	333.06	20.40	238.20	23.65	471.58	0.72	0.00	0.72	324.82	11.53	203.20	15.33	552.11

Cuadro 26: AM Best (20 % de restricciones) OpCruce: Uniform

	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	234.96	21.47	60.00	23.13	297.45	0.72	0.00	0.72	224.13	13.83	6.00	14.07	353.13
Seed 456	0.67	0.00	0.67	231.27	21.68	72.00	23.67	291.62	0.72	0.00	0.72	218.57	10.83	99.00	14.63	356.00
Seed 789	0.67	0.00	0.67	240.18	21.38	102.00	24.20	295.27	0.72	0.00	0.72	217.46	10.87	100.00	14.71	358.57
Seed 101112	0.67	0.00	0.67	228.95	19.61	126.00	23.09	286.91	0.72	0.00	0.72	217.10	13.83	6.00	14.07	355.17
Seed 131415	0.67	0.00	0.67	230.41	19.69	117.00	22.93	299.00	0.72	0.00	0.72	220.24	10.90	97.00	14.62	352.06
Media	0.67	0.00	0.67	233.15	20.77	95.40	23.41	294.05	0.72	0.00	0.72	219.50	12.05	61.60	14.42	354.99

Cuadro 27: AM Best (10 % de restricciones) OpCruce: BestFirst

	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	363.20	19.59	288.00	23.52	479.77	0.72	0.00	0.72	330.16	14.29	0.00	14.29	546.61
Seed 456	0.67	0.00	0.67	337.78	19.60	266.00	23.22	465.69	0.72	0.00	0.72	322.90	14.29	0.00	14.29	539.30
Seed 789	0.67	0.00	0.67	336.90	21.74	225.00	24.80	475.13	0.72	0.00	0.72	320.71	10.82	229.00	15.11	537.11
Seed 101112	0.67	0.00	0.67	334.30	22.15	136.00	24.01	479.15	0.72	0.00	0.72	329.53	14.29	0.00	14.29	554.16
Seed 131415	0.67	0.00	0.67	333.72	19.59	246.00	22.94	491.67	0.72	0.00	0.72	326.32	14.29	0.00	14.29	539.02
Media	0.67	0.00	0.67	341.18	20.53	232.20	23.70	478.28	0.72	0.00	0.72	325.92	13.59	45.80	14.45	543.24

Cuadro 28: AM Best (20 % de restricciones) OpCruce: BestFirst

	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	230.67	21.72	98.00	24.43	288.72	0.72	0.00	0.72	217.39	13.83	6.00	14.07	353.76
Seed 456	0.67	0.00	0.67	234.38	20.36	177.00	25.26	290.79	0.72	0.00	0.72	215.24	13.83	6.00	14.07	352.24
Seed 789	0.67	0.00	0.67	229.86	19.58	125.00	23.04	290.82	0.72	0.00	0.72	219.67	13.83	6.00	14.07	359.01
Seed 101112	0.67	0.00	0.67	236.69	19.70	127.00	23.21	294.39	0.72	0.00	0.72	219.21	13.83	6.00	14.07	361.37
Seed 131415	0.67	0.00	0.67	234.23	22.06	55.00	23.58	292.04	0.72	0.00	0.72	218.04	10.90	97.00	14.62	355.49
Media	0.67	0.00	0.67	233.16	20.69	116.40	23.91	291.35	0.72	0.00	0.72	217.91	13.25	24.20	14.18	356.37

Cuadro 29: AM Best (10 % de restricciones) OpCruce: BestLast

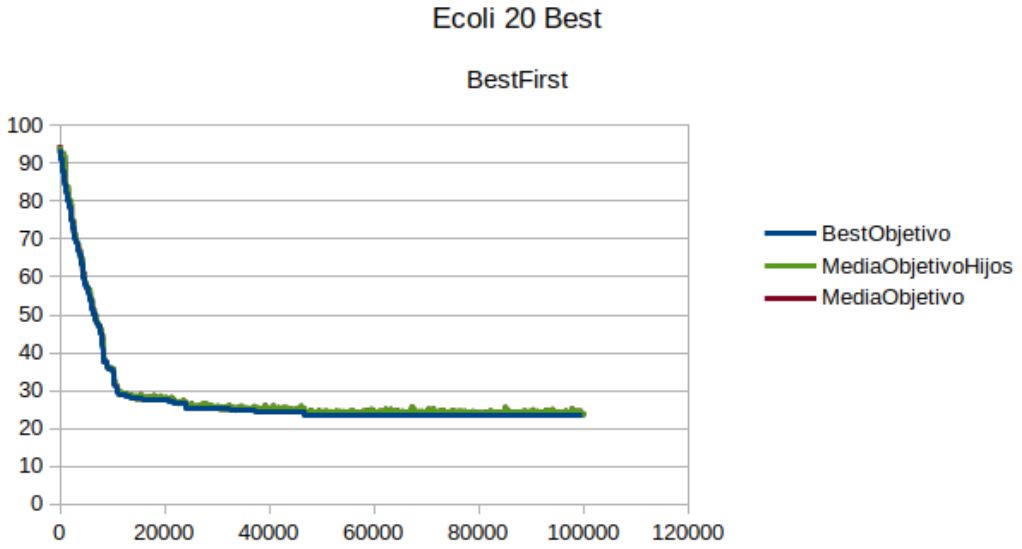
	Iris				Ecoli				Rand				NewThyroid			
	TasaC	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T
Seed 123	0.67	0.00	0.67	344.62	19.57	248.00	22.95	469.84	0.72	0.00	0.72	326.32	14.29	0.00	14.29	538.53
Seed 456	0.67	0.00	0.67	335.20	21.95	149.00	23.98	464.35	0.72	0.00	0.72	331.81	14.29	0.00	14.29	549.72
Seed 789	0.67	0.00	0.67	336.05	19.54	250.00	22.95	483.92	0.72	0.00	0.72	324.53	14.29	0.00	14.29	546.50
Seed 101112	0.67	0.00	0.67	336.82	19.64	239.00	22.89	467.61	0.72	0.00	0.72	321.29	14.29	0.00	14.29	542.12
Seed 131415	0.67	0.00	0.67	385.33	21.43	269.00	25.09	480.93	0.72	0.00	0.72	326.62	14.29	0.00	14.29	544.61
Media	0.67	0.00	0.67	347.60	20.43	231.00	23.57	473.33	0.72	0.00	0.72	326.12	14.29	0.00	14.29	544.30

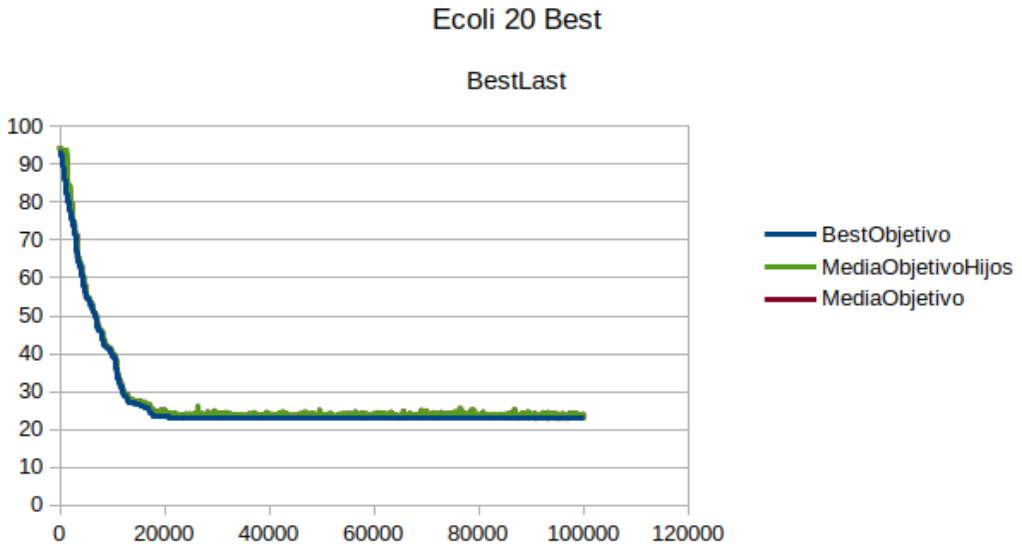
Cuadro 30: AM Best (20 % de restricciones) OpCruce: BestLast

En estos dado continua el patrón discutido anteriormente. El operador uniforme da mejor solución para 'Ecoli'. Ambos operadores de segmento fijo superan las soluciones con respecto al set 'NewThyroid'. Se predijo una mejoría del operador que favoreciese la exploración. Este operador supera a *BestFirst* en el set 'Ecoli'. Mientras que se queda muy cercano en 'NewThyroid'. La teoría de un menor número de óptimos locales podría explicar que *BestFirst* produzca mejores resultados (aunque por muy poco) que *BestLast*.

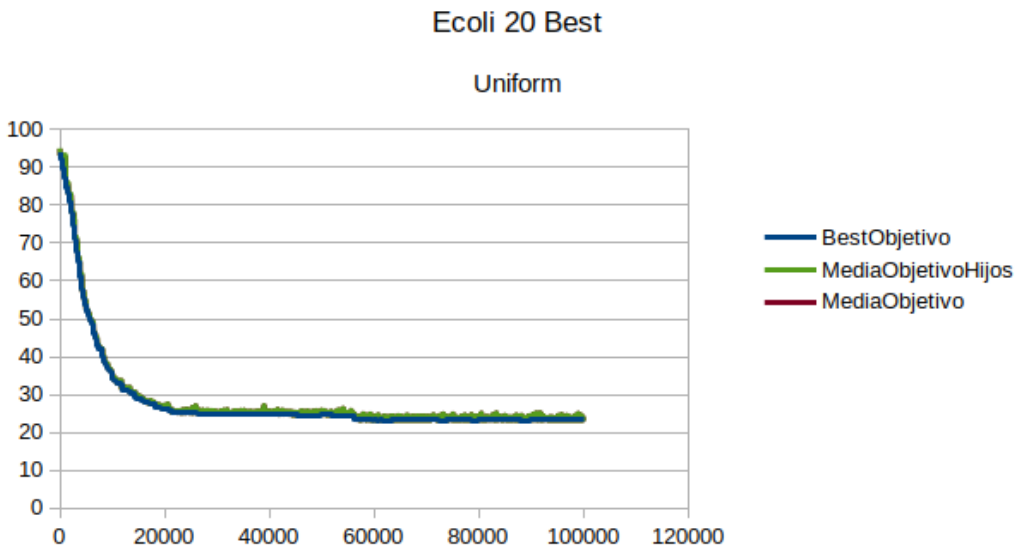
6.3.2.6. Convergencia en algoritmo memético *Best*

Se comienza viendo la convergencia entre las variaciones del operador de cruce de segmento fijo.





Y por último el cruce uniforme:



En este caso el grado de convergencia es muy similar en todos los operadores. Sin embargo se puede apreciar que *BestLast* converge durante más tiempo, provocando que llegue al óptimo local antes. De estas observaciones se pueden sacar bastante información. Que se observe el mismo grado de convergencia deja ver que la mayor parte de esta viene de las búsqueda local. Al elegir siempre los mejores elementos e intentar mejorarlos, ya que estos se conservan por el elitismo, provoca una gran convergencia al principio. Sin embargo, al acercarse a los óptimos locales, la búsqueda local comienza a ser más inefectiva. Se ve como *BestLast* consigue alcanzar el óptimo local antes ya que produce una mejor exploración del entorno. Esto le permitirá alcanzar un padre optimizable por la búsqueda local más fácilmente.

6.3.3. Problema de mutación

Luego de realizar el anterior estudio se descubre un pequeño problema en la implementación del código. Por la forma que se tenía implementado, la mutación no afectaba a la mejor solución de los hijos. Puede haber ocasiones en las que esta restricción sea necesaria. Sin embargo, en este caso ha modificado ligeramente los resultados. Al evitar cambios en la mejor solución de los hijos, se ha quitado parte del poder de exploración de este operador. Debido a la gran cantidad de tiempo que consume el ejecutar todas las pruebas, no es posible repetir el estudio completo. En cambio, se ejecutará una sola vez por cada variación de los algoritmos vistos. Se utiliza la semilla 123 para todas las ejecuciones y un 10 % de restricciones. Se muestra una tabla de comparación con los antiguos y nuevos resultados:

	Ecoli Old				Ecoli New				NewThyroid Old				NewThyroid New			
	TasaC	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T	*TasaC*	*Inf*	*Agr*	T
AGG-UN	21.78	81.20	24.03	830.51	21.24	92.00	23.78	727.57	11.81	70.40	14.51	407.29	13.83	6.00	14.07	355.33
AGG-SF(BF)	22.00	78.60	24.17	815.03	22.29	68.00	24.17	761.09	12.55	51.40	14.52	415.42	10.82	112.00	15.12	354.08
AGG-SF(BL)	24.14	99.20	24.14	818.28	23.85	139.00	27.69	739.66	14.43	47.80	14.43	407.35	10.82	113.00	15.15	353.64
AGE-UN	22.18	84.80	24.53	806.43	22.04	74.00	24.09	660.67	13.29	21.60	14.12	399.88	13.83	6.00	14.07	368.29
AGE-SF(BF)	22.56	87.60	24.98	812.05	21.91	81.00	24.15	658.17	12.14	63.20	14.56	401.82	13.83	6.00	14.07	329.94
AGE-SF(BL)	28.12	227.40	34.41	825.48	19.52	146.00	23.56	674.91	13.74	8.60	14.07	400.85	10.82	114.00	15.20	331.99
AM-(10,1.0)-UN	22.09	98.80	24.82	38.31	22.95	250.00	29.86	47.07	12.56	40.60	14.12	91.59	10.82	108.00	14.96	105.84
AM-(10,1.0)-SF(BF)	22.57	163.60	27.09	38.23	24.63	108.00	27.61	48.49	11.96	75.50	14.29	92.49	10.83	100.00	14.66	106.77
AM-(10,1.0)-SF(BL)	22.54	114.80	25.72	38.65	23.61	131.00	27.23	46.75	12.01	58.60	14.25	91.43	13.93	2.00	14.01	108.62
AM-(10,0.1)-UN	20.19	97.20	22.88	291.81	19.62	107.00	22.58	179.20	10.87	101.00	14.75	358.76	10.82	112.00	15.12	227.97
AM-(10,0.1)-SF(BF)	21.44	91.80	23.97	293.20	19.71	100.00	22.47	178.47	13.25	24.20	14.18	357.20	13.83	6.00	14.07	221.89
AM-(10,0.1)-SF(BL)	21.04	93.60	23.62	295.39	19.67	109.00	22.68	175.06	12.66	42.40	14.29	355.88	13.83	6.00	14.07	225.69
AM-(10,0.1mej)-UN	19.65	115.00	22.83	291.51	21.33	101.00	24.13	177.27	11.45	83.80	14.66	357.22	10.81	110.00	15.03	227.04
AM-(10,0.1mej)-SF(BF)	20.77	95.40	23.41	294.05	19.58	105.00	22.49	182.58	12.05	61.60	14.42	354.99	13.83	6.00	14.07	228.02
AM-(10,0.1mej)-SF(BL)	20.69	116.40	23.91	291.35	19.68	105.00	22.58	185.37	13.25	24.20	14.18	356.37	13.83	6.00	14.07	229.39

Cuadro 31: Comparación tras modificación

En esta tabla podemos comprobar los nuevos resultados obtenidos detenidamente. Como se ha comentado anteriormente, este nuevo cambio va a favorecer la exploración de soluciones mas variadas. Ya que favorece la exploración a partir de la mejor solución, esta será exploración con muy buena calidad. En los conjuntos de datos pequeños no se aprecian cambios. Sin embargo existen cambios en los medianos y grandes.

Se puede comparar los valores de la función objetivo para ver cuánto se han modificado las soluciones finales. Se ignoran los casos dónde la diferencia es mayor de cinco por ser considerados *outliers* causados por esta semilla en concreto. Aunque parezca que la modificación es muy pequeña, se puede apreciar que ha tenido cierto impacto en los algoritmos meméticos. Esto puede ser por añadir exploración a este algoritmo el cual se ha visto que converge rápidamente. Además provoca cierta mejoría general en el operador Uniforme y *BestFirst*. Ha provocado la mayor cantidad de variación en *BestLast*, esto puede deberse por la mayor exploración en un cruce que ya se concentra en esto. Este aumento de exploración puede provocar que a veces encuentre un camino para converger de forma eficiente. Mientras que otras veces explore demasiado y no converja lo suficientemente rápido.

Una ejecución de cada uno de estos algoritmos toma aproximadamente 8-10 horas. Se ha visto que la diferencia con las últimas modificaciones no es notable respecto a los datos del estudio. A pesar de esto, en el estudio se ha intentado mantener la distancia respecto a diferencias de datos exactas. Esto hace que las conclusiones de este estudio se puedan mantener. Sin embargo, para la tabla de resultados globales se comparará respecto a los anteriores algoritmos. Ya que esta modificaciones si que ha tenido algo de impacto positivo en las soluciones alcanzadas por los algoritmos meméticos se decide crear esta tabla con los resultados de esta modificación. Se dejará como trabajo posterior el realizar la media de las tablas con diferentes semillas para poder disminuir *outliers*.

6.4. Comparación Global

A continuación se exponen las tablas de comparación global de los algoritmos.Como se explicó anteriormente, se dejará la comparación a fondo entre las nuevas metaheurísticas para la siguiente práctica. Se entrará más en las diferencias de los algoritmos recién implementados con el algoritmo de comparación COPKM y la búsqueda local.

	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	0.67	0.10	32.82	2.00	32.87	1.02	0.76	0.00	0.76	0.09	14.29	0.00	14.29	0.16
BL	0.67	0.00	0.67	0.17	23.02	45.20	24.27	2.68	0.76	0.00	0.76	0.16	11.83	77.60	14.81	0.44
AGG-UN	0.67	0.00	0.67	202.24	21.24	92.00	23.78	727.57	0.72	0.00	0.72	190.88	13.83	6.00	14.07	355.33
AGG-SF(BF)	0.67	0.00	0.67	210.01	22.29	68.00	24.17	761.09	0.72	0.00	0.72	187.45	10.82	112.00	15.12	354.08
AGG-SF(BL)	0.67	0.00	0.67	205.49	23.85	139.00	27.69	739.66	0.72	0.00	0.72	189.76	10.82	113.00	15.15	353.64
AGE-UN	0.67	0.00	0.67	203.56	22.04	74.00	24.09	660.67	0.72	0.00	0.72	186.09	13.83	6.00	14.07	368.29
AGE-SF(BF)	0.67	0.00	0.67	198.33	21.91	81.00	24.15	658.17	0.72	0.00	0.72	184.76	13.83	6.00	14.07	329.94
AGE-SF(BL)	0.67	0.00	0.67	198.92	19.52	146.00	23.56	674.91	0.72	0.00	0.72	184.34	10.82	114.00	15.20	331.99
AM-(10,1.0)-UN	0.67	0.00	0.67	96.77	22.95	250.00	29.86	47.07	0.72	0.00	0.72	95.24	10.82	108.00	14.96	105.84
AM-(10,1.0)-SF(BF)	0.67	0.00	0.67	98.20	24.63	108.00	27.61	48.49	0.72	0.00	0.72	93.41	10.83	100.00	14.66	106.77
AM-(10,1.0)-SF(BL)	0.67	0.00	0.67	91.73	23.61	131.00	27.23	46.75	0.73	14.00	0.71	94.55	13.93	2.00	14.01	108.62
AM-(10,0.1)-UN	0.67	0.00	0.67	149.07	19.62	107.00	22.58	179.20	0.72	0.00	0.72	139.13	10.82	112.00	15.12	227.97
AM-(10,0.1)-SF(BF)	0.67	0.00	0.67	209.05	19.71	100.00	22.47	178.47	0.72	0.00	0.72	141.03	13.83	6.00	14.07	221.89
AM-(10,0.1)-SF(BL)	0.67	0.00	0.67	215.78	19.67	109.00	22.68	175.06	0.72	0.00	0.72	138.89	13.83	6.00	14.07	225.69
AM-(10,0.1mej)-UN	0.67	0.00	0.67	153.66	21.33	101.00	24.13	177.27	0.72	0.00	0.72	142.03	10.81	110.00	15.03	227.04
AM-(10,0.1mej)-SF(BF)	0.67	0.00	0.67	154.54	19.58	105.00	22.49	182.58	0.72	0.00	0.72	144.39	13.83	6.00	14.07	228.02
AM-(10,0.1mej)-SF(BL)	0.67	0.00	0.67	155.00	19.68	105.00	22.58	185.37	0.72	0.00	0.72	148.02	13.83	6.00	14.07	229.39

Cuadro 32: 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	0.67	0.12	30.43	0.00	30.43	1.41	0.76	0.00	0.76	0.12	14.29	0.00	14.29	0.22
BL	0.67	0.00	0.67	0.17	22.82	130.20	24.59	2.53	0.76	0.00	0.76	0.17	12.81	126.80	15.18	0.41
AGG-UN	0.67	0.00	0.67	278.53	21.91	156.00	24.03	1093.87	0.72	0.00	0.72	261.67	10.81	257.00	15.62	491.35
AGG-SF(BF)	0.67	0.00	0.67	273.06	21.73	164.00	23.96	1109.94	0.72	0.00	0.72	256.21	14.29	0.00	14.29	500.87
AGG-SF(BL)	0.67	0.00	0.67	278.53	21.91	156.00	24.03	1093.87	0.72	0.00	0.72	261.67	10.81	257.00	15.62	491.35
AGE-UN	0.67	0.00	0.67	258.73	24.72	162.00	26.92	971.16	0.72	0.00	0.72	253.35	14.29	0.00	14.29	456.48
AGE-SF(BF)	0.67	0.00	0.67	262.54	21.90	177.00	24.32	993.95	0.72	0.00	0.72	254.21	14.29	0.00	14.29	465.19
AGE-SF(BL)	0.67	0.00	0.67	272.76	22.16	175.00	24.54	977.44	0.72	0.00	0.72	249.89	14.29	0.00	14.29	465.96
AM-(10,1.0)-UN	0.67	0.00	0.67	110.53	21.20	318.00	25.53	64.02	0.72	0.00	0.72	111.91	14.29	0.00	14.29	131.40
AM-(10,1.0)-SF(BF)	0.67	0.00	0.67	108.41	22.90	168.00	25.19	61.66	0.72	0.00	0.72	111.20	14.29	0.00	14.29	133.85
AM-(10,1.0)-SF(BL)	0.67	0.00	0.67	114.13	22.86	147.00	24.05	61.25	0.72	0.00	0.72	111.37	10.82	229.00	15.11	141.25
AM-(10,0.1)-UN	0.67	0.00	0.67	283.63	21.81	205.00	24.60	253.66	0.72	0.00	0.72	187.18	14.29	0.00	14.29	308.54
AM-(10,0.1)-SF(BF)	0.67	0.00	0.67	311.96	19.45	235.00	22.65	273.11	0.72	0.00	0.72	184.58	14.29	0.00	14.29	308.37
AM-(10,0.1)-SF(BL)	0.67	0.00	0.67	201.44	25.88	158.00	23.85	264.49	0.72	0.00	0.72	191.66	10.87	231.00	15.19	318.31
AM-(10,0.1mej)-UN	0.67	0.00	0.67	200.40	19.59	301.00	22.99	265.33	0.72	0.00	0.72	189.36	14.29	0.00	14.29	313.25
AM-(10,0.1mej)-SF(BF)	0.67	0.00	0.67	200.06	21.46	216.00	24.40	272.23	0.72	0.00	0.72	189.88	14.29	0.00	14.29	313.64
AM-(10,0.1mej)-SF(BL)	0.67	0.00	0.67	212.07	20.89	232.00	24.05	268.92	0.72	0.00	0.72	190.91	14.29	0.00	14.29	321.64

Cuadro 33: 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. Esto provoca que no tome en cuenta otras posibles soluciones que incumplen mayor número de restricciones. Se aprecia como la mayoría de los algoritmos nuevos posee mayor

Infeasibility. Llegan a una solución con mayor *Infeasibility* pero una desviación notablemente menor, lo que le permite llegar a un mejor valor de la función objetivo. Sin embargo, el cambio más notable entre estos grupos de algoritmos es el tiempo consumido. El tiempo consumido varía notablemente. Comparado con la búsqueda local, los nuevos tiempos son [2400, 40000] % mayores. Esto puede ser inaceptable para algunos casos concretos que necesiten resultados en tiempo real. Sin embargo, también existen casos donde se dispone de un periodo de tiempo suficiente y la importancia de obtener la mejor solución prima.

Para conjuntos pequeños y fácilmente optimizables si que se podría usar la búsqueda local. Ya que parece alcanzar buenos óptimos locales con cierta facilidad y rapidez. Sin embargo, las técnicas basadas en poblaciones proveen la posibilidad de descubrir mejores soluciones con un mayor coste de tiempo. Esto puede ser prometedor en ocasiones donde no se esté limitado por el tiempo o se tenga acceso a nodos de cómputo lo suficientemente potentes.

6.5. 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. Los algoritmos genéticos suponen una mayor cantidad de exploración y por ello, mayor tiempo consumido. Se ha visto los diferentes grados de convergencia dependiendo de los esquemas de evolución y como estos cambian dependiendo de los diferentes operadores. Los algoritmos meméticos proporcionan una hibridación entre las dos técnicas, lo cual permite algoritmos más rápidos y que sigan obteniendo mejores soluciones que la simple búsqueda local. La importancia de la selección del conjunto al que aplicar la búsqueda local y como esto se diferencia de los algoritmos genéticos ha sido mostrada repetidas veces. También se ha graficado y analizado los posibles resultados de crear una cantidad elevada de generaciones y cuándo es recomendable hacerlo. Por último, la última modificación realizada en el programa ha servido como un estudio de posibles alteraciones en el programa. Aunque sean muy pequeñas, al crearse tantas generaciones, pueden variar de forma considerable los resultados finales.

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