

# BIOGEOGRAPHY BASED OPTIMIZATION

Pablo Huertas Arroyo

21 de junio de 2022



# UNIVERSIDAD DE GRANADA

Correo: [phuertas@correo.ugr.es](mailto:phuertas@correo.ugr.es)  
DNI:77033078Y  
Grupo 3A, subgrupo 2

# Índice

<b>1. RESUMEN</b>	<b>2</b>
1.1. Algoritmo BBO . . . . .	2
1.2. Problema donde se prueba el BBO . . . . .	4
1.3. Exploración vs Explotacion . . . . .	6
1.4. Equilibrio . . . . .	6
<b>2. ADAPTACION DEL BBO AL PROBLEMA DE LA MINIMA DIS- ERSION DIFERENCIAL</b>	<b>7</b>
2.1. Descripción de la función objetivo . . . . .	8
2.2. Descripción de los operadores comunes . . . . .	9
2.2.1. Generación de soluciones aleatorias . . . . .	9
<b>3. PROPUESTA DE MEJORA DE LA METAHEURISTICA</b>	<b>10</b>
3.1. Hibridacion con Enfriamiento Simulado . . . . .	10
3.1.1. Algoritmo de Enfriamiento Simulado . . . . .	10
3.1.2. Operador de mutacion . . . . .	12
3.2. Hibridacion con Busqueda Local . . . . .	13
3.2.1. Busqueda Local . . . . .	13
<b>4. ESTUDIO EXPERIMENTAL Y ANÁLISIS DE RESULTADOS</b>	<b>16</b>
4.1. Tabla resumen . . . . .	18
4.2. Analisis resultados BBO . . . . .	19
4.2.1. BBO sin hibridacion . . . . .	19
4.2.2. BBO con hibridacion BL . . . . .	19
4.2.3. BBO con hibridacion BL++ . . . . .	19
4.2.4. BBO con hibridacion ES . . . . .	20

## 1. RESUMEN

### 1.1. Algoritmo BBO

El algoritmo BBO (Biogeography Based Optimization) es un algoritmo evolutivo que optimiza una función de forma estocástica e iterativa mejorando las soluciones candidatas con respecto a una medida de calidad llamado fitness.

BBO optimiza un problema manteniendo una población de soluciones candidatas (elitismo), y creando nuevas soluciones con dos procesos llamados migración y mutación que se encargan de mejorar las soluciones nuevas generadas. Estas nuevas soluciones serán candidatas si son de buena calidad.

La Bio-Geografía es el estudio de la extinción y la distribución geográfica de las especies biológicas, cuyos modelos matemáticos describen la evolución de nuevas especies, la migración de especies entre islas vecinas y la extinción de las mismas. Para poder comprender el comportamiento de la metaheurística BBO es necesario conocer definiciones y aspectos fundamentales que serán primordiales en el proceso de desarrollo de este algoritmo. A continuación se presenta una revisión de ciertos factores tales como, los índices de Migración ( $\lambda$ ,  $\mu$ ), el índice de adecuación de un hábitat HSI (Habitat Suitability Index), y las variables de idoneidad SIVs (Suitability Index Variables), los cuales son las bases de este método de optimización.

Un área geográfica que se adapta mejor como residencia para albergar especies biológicas se dice que es un hábitat que tiene un alto índice de adecuación (HSI), por lo tanto, un hábitat que tiene un alto HSI puede estar compuesto por una gran diversidad de recursos, los cuales pueden ser, cascadas, diversidad topográfica, diversidad vegetativa, áreas de tierra, ríos, lagos, etc. De esta manera, surge un nuevo término denominado variables de idoneidad (SIVs), que son variables independientes que representan las características de habitabilidad de una isla.

Aquellos hábitats que tienen un alto HSI son capaces de hospedar a un mayor número de especies que aquellos hábitats que tienen un bajo HSI; de la misma forma se tiene que los índices de migración (inmigración y emigración) varían con respecto al HSI. Así podemos determinar que aquellos hábitats con un alto HSI tienen un alto índice de emigración debido principalmente al número de especies que el hábitat puede tener, mientras que el índice de inmigración resulta bajo ya que el hábitat puede contener demasiadas especies como para albergar otras provenientes de islas vecinas. Esto puede ocasionar que las nuevas especies no sobrevivan debido a que existiría una gran competencia por los recursos.

Por otro lado, el índice de inmigración en hábitats que tienen un bajo HSI es alto debido fundamentalmente a que este tipo de hábitats tienen una población pequeña, por lo tanto, disponen de áreas libres que podrían albergar nuevas especies. Hay que resaltar que el alto índice de inmigración ocurre por lo antes expuesto, más no porque especies de islas vecinas quieran llegar a estos hábitats, ya que después de to-

do, las características que tienen estos hábitats son indeseables para nuevas especies.

$\lambda$  representa el índice de inmigración, mientras que  $\mu$  representa el índice de emigración, ambos índices se encuentran en función del número de especies de una isla,  $S_o$  representa el punto de equilibrio donde  $\lambda$  es igual a  $\mu$ , y  $S_{max}$  representa el máximo número de especies que la isla puede soportar.

Si observamos la curva de inmigración representado por  $\lambda$  se puede notar que la tasa máxima de inmigración representada por  $I$ , ocurre cuando no existen especies en la isla, y a medida que el número de especies va incrementando, la tasa de inmigración va disminuyendo, debido a que la isla se va llenando, y cada vez menos especies sobrevivirán al proceso de inmigración; por otro lado, se verifica también que cuando una isla alberga el máximo número de especies que esta puede soportar ( $S_{max}$ ) la tasa de inmigración es cero.

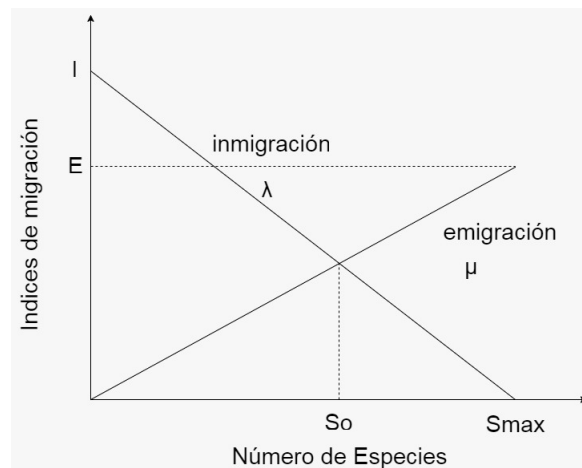


Figura 1: Curva de emigración/inmigración

Ahora consideremos la curva de emigración  $\mu$ , en donde si no existen especies en la isla la tasa de emigración es cero, y mientras el número de especies en la isla va incrementando la tasa de emigración también lo hace, lo cual quiere decir que a medida que aumenta el número de especies en la isla, más especies están disponibles para emigrar a islas vecinas. La tasa máxima de emigración  $E$ , ocurre cuando la isla alberga al máximo número de especies que esta puede soportar.

En resumen, se presentan una serie de diferentes factores que serán las bases de dicho método de optimización

- **Fitness:** Se utiliza para calcular la calidad de las soluciones. (En nuestro caso la dispersión)
- $\mu$  : Índice de emigración de una población en concreto. Esta variable indica cuanto probable va a ser que algún elemento de dicha solución emigre a otra.
- $\lambda$  : Índice de inmigración de una población en concreto. Esta variable indica cuanto probable va a ser que algún elemento de dicha solución sea reemplazado

por un elemento de otra solución.

- **Isla** : En nuestro caso se trata de una población
- **HSI** : Habitat Suitability Index. índice numérico que representa la capacidad de un hábitat de mantener un cierto nivel de población de diferentes especies. Se obtiene de la combinación de las diferentes variables que afectan a la calidad de vida de las especies que habitan ese hábitat.
- **SIV** : Suitability Index Variables. Variables independientes que representan las características de habitabilidad de una isla. (En nuestro caso cada SIV será cada una de las soluciones de la población)

Existen tres mecanismos principales de este algoritmo, los cuales son:

- **Elitismo** : Antes de generar nuevas soluciones se guardan un conjunto de soluciones que sean las mejores. Estas soluciones serán introducidas al finalizar por las peores de las nuevas generadas.
- **Migración** : Se realiza una migración de una solución a otra. Por probabilidad los elementos de las mejores soluciones se migran a otras soluciones.
- **Mutación** : Se realiza una mutación de una solución. Por probabilidad se cambia un elemento de una solución por otro. Se debe conservar la factibilidad de las soluciones.

## 1.2. Problema donde se prueba el BBO

El problema elegido a abordar en esta práctica es el siguiente: Problema de la mínima dispersión diferencial (**MDD**). Es un problema de optimización combinatoria consistente en seleccionar un subconjunto  $M$  de  $m$  elementos ( $|M|=m$ ) de un conjunto inicial  $N$  de  $n$  elementos (con  $n>m$ ) de forma que se minimice la dispersión entre los elementos escogidos.

Este problema tiene diferentes **aplicaciones en el campo de la optimización**, como pueden ser la elección de la localización de elementos públicos, selección de grupos homogéneos, identificación de redes densas, reparto equitativo, problemas de flujo, etc

$$\begin{aligned} \text{Minimize} \quad & \max_{i \in M} \left\{ \sum_{j \in M} d_{ij} \right\} - \min_{i \in M} \left\{ \sum_{j \in M} d_{ij} \right\} \\ \text{Subject to} \quad & M \subset N, |M| = m \end{aligned}$$

donde:

- $M$  es una solución al problema que consiste en un vector binario que indica los  $m$  elementos seleccionados
- $d_{ij}$  es la distancia existente entre los elementos  $i$  y  $j$ .

Para resolver este problema se utilizarán 50 casos seleccionados con distancias reales con,  $n$  entre 25,50,75,100,125,150, y  $m$  entre 2 y 45.

La Dispersión de una Solución es la diferencia de los valores extremos, es decir, la diferencia de la sumas de las distancias de dichos puntos al resto de los puntos. Por ejemplo, si tenemos 8 puntos para colocar farmacias, y solo podemos colocar 4, **¿cuál es la forma de colocarlas, de forma que se reduzca la dispersión?**

Los datos se encuentran en unos ficheros *.txt*, donde hay una primera línea que indica el numero de elementos  $n$  y el número de elementos a seleccionar  $m$  del problema.

Luego se encuentran  $n*(n-1)/2$  líneas con el formato  $i,j,d_{ij}$  que tienen el contenido de las **distancias entre los elementos**.

En mi caso, para los dos algoritmos he leído estos ficheros y he almacenado los datos en una matriz distancias completa, donde la diagonal es 0, y las triangulares superiores e inferiores son simétricas entre sí.

La posición (2,3) de la matriz distancias es la distancia entre los elementos 2 y 3, que a su vez es la misma que la posición (3,2).

La **representación de la solución** es un vector de enteros, donde la posición  $i$ -ésima es el numero del elemento que esta seleccionado. Mantengo en el conjunto de datos solución en la implementación el vector binario usado en la practica anterior, para la reutilizacion de código.

Para la **factorización de la función objetivo**, a la hora de generar una nueva solución no es necesario volver a calcular por completo el vector de distancias para obtener la nueva dispersión. Basta con restar la distancia a cada elemento de la solución al elemento que se ha quitado de la solución actual, y sumarle la distancia del nuevo elemento a todas las demás de la solución.

Entonces,teniendo el vector de distancias actualizado, para saber la dispersión de dicho conjunto de elementos restamos la mayor distancia de dicho vector con la menor

Debido a que se requiere aleatoriedad en ambos algoritmos, ya que son probabilísticos, he usado un vector de semillas, donde en cada iteración que realiza cada algoritmo se genera una nueva semilla, y se utiliza para generar nuevas soluciones. El valor estático de la semilla sirve para que cada vez que se ejecute el algoritmo, se obtengan las mismas Soluciones.

También se pedía calcular el tiempo de ejecución de cada algoritmo, por lo que he usado objetos de la clase **<chrono>** para tener una alta precisión en los tiempos, y los muestro en **segundos**.

Al finalizar cada algoritmo calculo el tiempo demorado por dicho algoritmo y la dispersión de la mejor solución encontrada.

### 1.3. Exploración vs Explotación

En BBO no hay algo como una operación de cruce; las soluciones se ajustan finamente gradualmente mientras el proceso continúa por la operación de migración. Esto da una ventaja al BBO sobre otras técnicas evolutivas. BBO tiene una gran habilidad de explotación en un problema de optimización global.

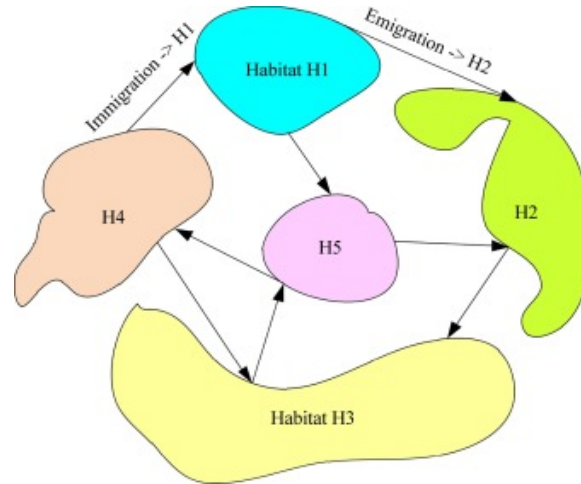


Figura 2: Islas

Para la explotación se van a utilizar las migraciones, que van a consistir en migrar elementos de las mejores soluciones a las peores. Esto se hace para que las soluciones buenas mejoren a las peores. Para la exploración se utilizan las mutaciones, que son las que se aplican a todas las soluciones en igual proporción. Habrá una variable que define con qué probabilidad se produce una mutación. Las mutaciones producen un cambio en la solución, y se debe buscar un elemento nuevo que va a ser el que entre en la solución por el anterior.

### 1.4. Equilibrio

Este algoritmo presenta muy buen equilibrio entre la exploración y la explotación. Esto se debe a que casi con igual proporción se van a realizar las migraciones y las mutaciones. Sin embargo, en el problema donde ha sido aplicado el algoritmo la migración no funciona tan bien como en otros casos, y esto se debe a que para ser factible la solución como hemos comentado antes debe tener un número exacto de elementos seleccionados. Por lo tanto, cuando se realiza la migración en una solución, se debe eliminar un elemento de los seleccionados de dicha solución y esto se hace de forma aleatoria. Se pueden perder elementos de las soluciones que sean prometedores por la aleatoriedad del algoritmo a la hora del reemplazamiento del elemento de la solución.

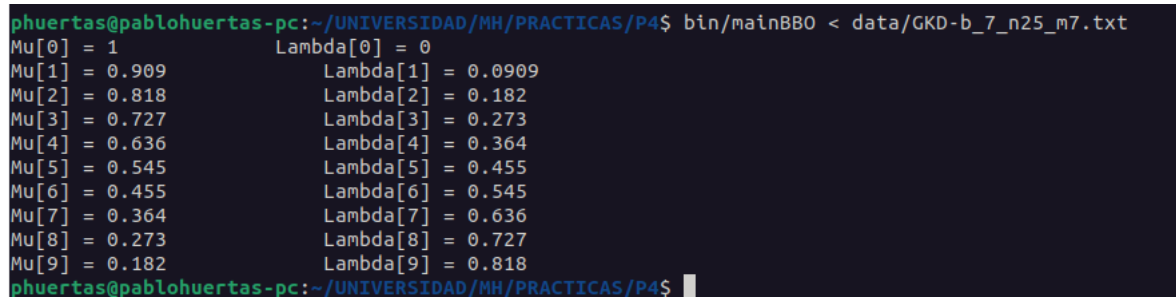
En este problema en específico encontramos una mejor explotación que exploración, ya que la exploración realmente no cambia mucho la solución, al tener que encontrar otro elemento que entre en esta para guardar la factibilidad. En cambio en la explotación a pesar de que podamos perder un elemento bueno de la solución, entrará un elemento que probabilísticamente debe ser prometedor también.

## 2. ADAPTACION DEL BBO AL PROBLEMA DE LA MINIMA DISPERSION DIFERENCIAL

Para adaptar este algoritmo al problema de la minima dispersion diferencial he tenido que hacer ciertos cambios.

Primero el tamaño de la poblacion(numero de islas), viene dado por el tamaño de la poblacion aleatoria generada.

Las variables de  $\mu$  y  $\lambda$  vienen dadas en un vector del mismo tamaño de la poblacion. Por ejemplo para una poblacion de tamaño 10 estos serian los valores de  $\mu$  y  $\lambda$ :



```
phuertas@pablohuertas-pc:~/UNIVERSIDAD/MH/PRACTICAS/P4$ bin/mainBBO < data/GKD-b_7_n25_m7.txt
Mu[0] = 1          Lambda[0] = 0
Mu[1] = 0.909      Lambda[1] = 0.0909
Mu[2] = 0.818      Lambda[2] = 0.182
Mu[3] = 0.727      Lambda[3] = 0.273
Mu[4] = 0.636      Lambda[4] = 0.364
Mu[5] = 0.545      Lambda[5] = 0.455
Mu[6] = 0.455      Lambda[6] = 0.545
Mu[7] = 0.364      Lambda[7] = 0.636
Mu[8] = 0.273      Lambda[8] = 0.727
Mu[9] = 0.182      Lambda[9] = 0.818
phuertas@pablohuertas-pc:~/UNIVERSIDAD/MH/PRACTICAS/P4$
```

Figura 3: Valores de  $\mu$  y  $\lambda$  para un tamaño de poblacion de 10

Ahora toca ordenar la poblacion por fitness, y esto se realiza ordenando de menor a mayor dispersion.

Las soluciones con menor dispersion estarán mas arriba en la poblacion. Asi se cumple la relacion 1 a 1 entre el vector de  $\mu$  y  $\lambda$  y la poblacion.

Es por esto, que la primera solucion será la mejor, y coincide con el máximo valor de  $\mu$  que es 1.

Es interesante que esta solucion al ser la mejor comparta sus características a las peores, que tendrán un valor de  $\lambda$  elevado.

Se genera un conjunto de elites, de tamaño elegido arbitrariamente, por ejemplo  $0.1 \cdot \text{tamaño de la poblacion}$  si la poblacion es minimo de 10 soluciones, o mayor sera este valor si la poblacion es mas pequeña para asegurarnos que hay al menos algun elite que se guarda.

Ademas vamos a ir almacenando siempre la mejor solucion encontrada. Cada vez que se ordena la poblacion por fitness, se comprueba si el primer elemento tiene menor dispersion que la mejor encontrada hasta el momento y si es asi se sustituye. Esta solucion sera la solucion final.

Luego se entraria en el bucle donde se generan cada una de las generaciones del algoritmo, con dos bucles internos correspondientes anidados, uno de 0 hasta el tamaño de la poblacion y otro de 0 hasta el numero de elementos que han de ser seleccionados en el caso del problema en concreto.

En el bucle más interno es donde se realiza el proceso de migracion, mientras que en el bucle intermedio es donde se va a realizar la mutacion con una probabilidad tambien elegida arbitrariamente.



El proceso de mutacion se podria meter dentro del bucle más interno, pero habria que disminuir la probabilidad de mutacion ya que se ejecutarían muchas más mutaciones de las que deseamos.

De todas formas esto es un parametro muy arbitrario, ya que el valor que asignemos a la probabilidad de mutacion va a depender mucho de si queremos una mayor explotacion o menor.

Cada vez que se migra o muta un elemento, se calcula la funcion objetivo de la solucion modificada.

---

**Algorithm 1:** Algoritmo de Optimizacion Basada de Biogeografia

---

**Input:**  $n, m, poblacion, prob_{mutacion}, n_{generaciones}, n_{elites}$

**Output:**  $solucion$

```

1   $poblacion \leftarrow (ORDENAR \text{ POR } FITNESS)$ 
2   $mu \leftarrow \{1, \dots, 0\}$ 
3   $lambda \leftarrow \{0, \dots, 1\}$ 

4   $mejor_{solucion} \leftarrow poblacion[0]$ 

5   $VectorDistancias \leftarrow GenerarVectorDistancias()$ 
6   $DispersionComparacion \leftarrow CalcularDispersion(VectorDistancias)$ 

7   $Mejora \leftarrow \text{TRUE}$ 

8  for  $n_{iteraciones} \in n_{generaciones}$  do
9       $elites \leftarrow \{poblacion[0], \dots, poblacion[n_{elites}]\}$ 

10     for  $i \in Size(poblacion)$  do
11         for  $j \in m$  do
12             if  $Random.Get(0, 1) \leq lambda[i]$  then
13                  $poblacion[i] \leftarrow \text{MIGRACION}$ 
14                  $poblacion[i] \leftarrow \text{RECALCULAR } FUNCION \text{ OBJETIVO}$ 
15             if  $Random.Get(0, 1) \leq prob_{mutacion}$  then
16                  $poblacion[i] \leftarrow \text{MUTACION}$ 
17                  $poblacion[i] \leftarrow \text{RECALCULAR } FUNCION \text{ OBJETIVO}$ 

18      $poblacion \leftarrow (ORDENAR \text{ POR } FITNESS)$ 
19      $poblacion \leftarrow (SUSTITUIR \text{ PEORES POR } ELITES)$ 

20     if  $mejor_{solucion}.dispersion < poblacion[0].dispersion$  then
21          $mejor_{solucion} \leftarrow poblacion[0]$ 

22 return  $\{mejor_{solucion}\}$ 

```

---

## 2.1. Descripción de la función objetivo

La función objetivo de este problema es la de encontrar la dispersión a partir de un vector de booleanos donde la posición  $i$ -ésima es 1 si el elemento  $i$ -ésimo está seleccionado, y 0 en caso contrario.

Para evaluar la función objetivo, se convierte internamente el vector de booleanos en una selección de elementos de números enteros.

Para ello, se recorre el vector de booleanos, y si la posición  $i$ -ésima es 1, se añade al final del vector de seleccionados el elemento  $i$ -ésimo.

Tenemos la matriz de distancias comentada anteriormente, y la selección de elementos, por lo que para evaluar la función objetivo, para cada elemento del vector de seleccionado, en la posición  $i$ -ésima del vector distancias, añadimos la distancia del elemento  $i$ -ésimo a todos los demás elementos del vector de seleccionados.

Las posiciones se corresponden 1 a 1 en los vectores de seleccionados y distancias.

---

**Algorithm 2:** Algoritmo de Evaluación de la Función Objetivo

---

**Input:** distancias(vector), seleccionados(vector), m(matriz distancias)  
1  $distancias \leftarrow 0$

2  $VectorDistancias \leftarrow GenerarVectorDistancias()$   
3  $DispersionComparacion \leftarrow CalcularDispersion(VectorDistancias)$

4  $Mejora \leftarrow \text{TRUE}$   
5 **for**  $i \in Size(seleccionados)$  **do**  
6      $acomparar \leftarrow i$   
7     **for**  $j \in Size(seleccionados)$  **do**  
8         **if**  $seleccionados[i] \neq acomparar$  **then**  
9              $distancias[i] += m[acomparar][seleccionados[i]]$

---

## 2.2. Descripción de los operadores comunes

Hay ciertos operadores y funciones que son comunes para los algoritmos desarrollados en esta práctica, ya que por ejemplo la generación de soluciones aleatorias es común y varios operadores más, por lo que voy a desglosar uno a uno para entrar más en profundidad.

### 2.2.1. Generación de soluciones aleatorias

Para la generación de la primera solución aleatoria, utilizo una función para generar soluciones aleatorias, donde el número de posibles elementos a escoger es  $emphn$  y el número que finalmente son seleccionados son  $emphm$ .

---

**Algorithm 3:** Algoritmo de Generación de soluciones Aleatorias

---

**Input:**  $n$ (número de puntos)  $m$ (número de puntos a seleccionar), semilla(número que simboliza una semilla estática)  
**Output:** solución(vector de booleanos)

1  $solucin \leftarrow \emptyset$   
2  $seleccionados \leftarrow \emptyset$

3 **while**  $Size(seleccionados) < m$  **do**  
4      $seleccionados \leftarrow \text{Numero aleatorio que no esta en seleccionados}$   
5      $solucin \leftarrow seleccionados.back$

---

### 3. PROPUESTA DE MEJORA DE LA METAHEURISTICA

Como propuesta de mejora del algoritmo BBO, he decidido usar un operador de reparacion de las soluciones, ya que como hablamos antes hay soluciones que al migrar o al mutar pueden perder la factibilidad. Este operador de reparacion tiene como objetivo no volver a hacer factible la solucion de forma aleatoria tal y como estabamos haciendo, sino que tener en cuenta al eliminar el o los elementos que sobran o introducir los que faltan teniendo en cuenta cuanto se aleja su fitness(dispersion) a la media de las dispersiones de la solucion actual.

Es decir, el eliminar elementos de la solucion se eliminaran los que tengan un fitness mas alejado de la media de fitness de dicha solucion, e igual al introducir un elemento, que se tendra en cuenta con que valor queda la media de fitness con ese elemento en la solucion. Por lo tanto para introducir un elemento se introducira el que menos aumente la dispersion (o la reduzca), mientras que al eliminar un elemento bastará con eliminar el que tenga el fitness mas alejado de la media de fitness de la solucion(ya sea por encima o por debajo).

#### 3.1. Hibridacion con Enfriamiento Simulado

Una tecnica de mejora del algoritmo es la Hibridacion con Enfriamiento Simulado.

Este algoritmo se lo aplico a las soluciones que pueden estar mas estancadas en un optimo local, ya que sabemos que el algoritmo de Enfriamiento simulado tiene buenas tecnicas de escape de estos optimos. Por lo tanto lo aplico por defecto a todas las soluciones cada 5 generaciones para que haya una exploracion mas pronunciada.

A continuacion se muestra una descripcion del algoritmo de Enfriamiento Simulado

##### 3.1.1. Algoritmo de Enfriamiento Simulado

El algoritmo de Enfriamiento Simulado (*Simulated Annealing*) es un algoritmo de búsqueda metaheurística para problemas de optimización global.

El objetivo de este algoritmo es encontrar una solución optima o casi optima de un problema en un espacio de búsqueda grande.

Tiene un criterio probabilístico de aceptacion de soluciones basado en Termodinámica.

La forma que tiene de escapar de óptimos locales, es la posibilidad de aceptar soluciones peores con una cierta probabilidad, la cual va disminuyendo conforme se va avanzando en el algoritmo hacia una buena solución.

Tiene una filosofía de diversificar al principio e intensificar al final, es decir, al principio del algoritmo se evaluan multiples soluciones distintas y se selecciona la mejor, y al final se intensifica la búsqueda explotandola.

El máximo de éxitos que se podrán generar como máximo en cada iteracion del algoritmo serán  $n$ .

El máximo de vecinos que se podrán generar como máximo en cada iteracion del algoritmo serán  $10 \cdot n$ .

La constante  $\mu$  tendra valor 0.3 en toda la ejecucion

La temperatura inicial se calculará como...

$$T_0 = \frac{\mu * C(S_0)}{-\ln(\varphi)} \quad (1)$$

siendo  $\varphi = \mu$

Beta se calculará de la forma...

$$\beta = \frac{t_i - t_f}{t_i * t_f * n} \quad (2)$$

siendo  $t_i$  la temperatura inicial,  $t_f$  la temperatura final y  $n$  el tamaño de la solución.

Al final de cada iteracion se calcula la temperatura que se va a tomar como nueva, y esta se calcula de la forma...

$$t_{k1} = \frac{t_k}{1 + (\beta * t_k)} \quad (3)$$

siendo  $t_k$  la temperatura actual y  $k$  el numero de iteracion.

El número máximo de evaluaciones de la funcion objetivo del algoritmo completo serán **100000**

---

**Algorithm 4:** Algoritmo de Enfriamiento Simulado

---

**Input:** TemperaturaInicial(temperatura inicial), TemperaturaFinal(temperatura final)  
**Output:** TemperaturaActual(temperatura actual)

```

1  TemperaturaInicial ← CalcularTemperaturaInicial( $\mu, \varphi, C(S_0)$ )
2  TemperaturaActual ← TemperaturaInicial
3  TemperaturaFinal ←  $10^{-3}$ 
4  nEnfriamientos ←  $1000/n$ 
5  maxVecinos ←  $10 * n$ 
6  maxExitos ←  $n$ 

7  iteraciones ← 0
8  evaluaciones ← 0
9   $s_o \leftarrow \text{CalcularsoluciónAleatoria}$ 
10  $s_{mejor} \leftarrow S_o$ 

11 while (TemperaturaActual > TemperaturaFinal) && (iteraciones < nEnfriamientos) && (nEvaluaciones < 100000) do
12   exitos ← 0
13   for  $i \in 1 \dots \text{maxVecinos}$  && exitos < maxExitos do
14      $S_{vecino} \leftarrow \text{GenerarsolucionVecinaAleatoria}()$ 
15     evaluaciones ← evaluaciones + 1

16      $\Delta_{dispersion} \leftarrow dispersion(S_{vecino}) - dispersion(S_{actual})$ 
17     Calculamos la probabilidad de que se acepte la nueva si es peor que la actual
18     probabilidad ←  $e^{-\Delta_{dispersion}/TemperaturaActual}$ 

19     if ( $\Delta_{dispersion} < 0$ ) or ( $\text{GetRandomNumber}(0, 1) < \text{probabilidad}$ ) then
20        $S_{actual} \leftarrow S_{vecino}$ 
21       exitos ← exitos + 1

22     if  $dispersion(S_{actual}) - dispersion(S_{mejor})$  then
23        $S_{mejor} \leftarrow S_{actual}$ 

24   beta ← CalcularBeta
25   TemperaturaActual ← CalcularTemperaturaActual( $t_k, \beta, t_i, n$ )
26   iteraciones ← iteraciones + 1
27 return  $S_{mejor}$ 

```

---

**3.1.2. Operador de mutacion**

La mutacion consiste en modificar con cierta probabilidad uno o varios genes de la poblacion (en este caso de una solución) aleatoriamente. La probabilidad de mutacion es dada por la constante *probabilidad 0.1*. Cuando muta un gen de un cromosoma, tenemos que encontrar otro gen del mismo cromosoma con el valor contrario, para mantener la factibilidad de la solución de dicho cromosoma. Por ejemplo, en una solución con 10 elementos donde se seleccionan 3, si se va a mutar el segundo seleccionado, tenemos que buscar uno de los 7 elementos que no esten seleccionados de manera aleatoria y cambiar el valor de cada gen. El rango de elementos que pueden ser mutados, van desde 0 hasta el producto del numero de cromosomas por el numero de genes por cromosoma.

Si la poblacion tiene 10 cromosomas, y cada cromosoma 5 genes, si se genera para mutar el elemento 15, será el sexto gen del segundo cromosoma.

---

**Algorithm 5:** Operador de Mutación

---

**Input:** p(poblacion), prob(probabilidad)**Output:** pnueva(poblacion generada)

```

1 rango mutacion ← p.NumeroDeCromosomas() · p.NumeroDeGenesPorCromosoma()
2 for i ∈ Size(p) do
3   if GenerarNumeroAleatorioEntre(0,1) < prob then
4     Genero aleatoriamente un elemento en el rango de mutacion
5     posicion ← GenerarNumeroAleatorioEntre(0,rango)
6     Para el elemento de la posicion generada, busco otro gen del mismo cromosoma con el valor contrario
7     posicion2 ← Gen del mismo cromosoma aleatorio con valor contrario
8     Swap(posicion, posicion2)
9 return pnueva
```

---

En la ejecucion del algoritmo se realizan 10 iteraciones y cada BL como máximo hara **10000** evaluaciones o no mejore la solución en todo el entorno. El valor usado para el numero de genes a mutar de la solución es  $t = 0.3$

### 3.2. Hibridacion con Busqueda Local

Anteriormente hemos visto una posible hibridacion con el algoritmo de Enfriamiento Simulado.

Ahora vamos a ver una hibridacion con la busqueda local. Sabemos que el algoritmo de Busqueda Local tiene una explotacion muy pronunciada, por lo que si lo combinamos con un algoritmo con buena exploracion como es el BBO en nuestro caso (que hemos visto que para este problema la explotacion no es su punto fuerte por la propia estructura de la funcion objetivo) podemos obtener muy buenos resultados y vamos a experimentar sobre ello.

A continuacion se muestra una descripcion del algoritmo de busqueda local.

#### 3.2.1. Busqueda Local

Este algoritmo es un tipo de algoritmos de busqueda por trayectorias simples. En este algoritmo, se empieza con una solución inicial completa y aleatoria, es decir, una Solución con  $M$  elementos que no se repiten entre sí. El orden de estos elementos no es relevante.

La idea es tras haber generado una completa Solución aleatoria válida, generar el **vecindario completo** de la Solución actual, **desordenarlo aleatoriamente**, y recorrerlo comparando en cada iteracion si se mejora la Dispersión.

Si se mejora la Dispersión, se **selecciona dicha Solución como Solución actual** y se vuelve a generar el vecindario. Este proceso se hace hasta que no se mejore la Dispersión con todo el vecindario generado o hasta que se hayan hecho **100000 evaluaciones de la funcion objetivo**. Es decir, comprobar 100000 veces si se mejora la Dispersión.

Como vemos este algoritmo se parece a Greedy en que ambos cuando encuentran una Solución mejor que la anterior la seleccionan, y no se espera en este caso a recorrer todo el vecindario para encontrar una mejor Solución. Es por eso que este algoritmo se llama Busqueda Local de **Primero el mejor**

La generación de la primera Solución aleatoria se hace con un bucle que va generando numeros aleatorios entre 0 y  $n-1$ , de forma que si no se ha añadido aún a la Solución, lo añade. Este proceso se repite hasta que el numero de elementos de la Solución sea igual a  $M$

Para la generación de vecinos, uso un vector de tuplas, que contienen el elemento que se va a intercambiar y el elemento que se va a intercambiar y va a entrar a la Solución provisional.

Por ejemplo, si tengo  $M=6$  y  $N=3$ , Solución provisional=(1,3,5), y genero el vecindario de esta Solución, este será el vector de tuplas

(1,0), (1,2), (1,4), (3,0), (3,2), (3,4), (5,0), (5,2), (5,4).

Entonces, desordena este vector aleatoriamente y se va intercambiando la posicion primera de la tupla que se encuentra en la Solución por la segunda posicion de la tupla que no se encuentra en la Solución

La factorización es la misma que en el algoritmo greedy, cuando se intercambia un elemento de la Solución por otro, en el vector distancias a cada elemento se le resta la distancia con el elemento que se elimina, y se le suma la distancia con el elemento

que se añade, además de añadir en la posición del elemento añadido la distancia con todos los demás de la Solución.

## PSEUDOCÓDIGO DEL ALGORITMO DE BÚSQUEDA LOCAL

---

**Algorithm 6:** Algoritmo de búsqueda local

---

```

1   $v \leftarrow 0, w \leftarrow 0$ 
2   $S \leftarrow D$ 
3   $T \leftarrow \emptyset$ 
4   $solucin \leftarrow \emptyset$ 
5   $Elementosrestantes \leftarrow V$ 
6   $DispersionComparacion \leftarrow \emptyset$ 
7   $Distancias \leftarrow \emptyset$ 
8   $Dispersion \leftarrow \emptyset$ 

9   $Copiasolucin \leftarrow \emptyset$ 
10  $CopiaDistancia \leftarrow \emptyset$ 
11  $Vecindario \leftarrow \emptyset$ 

12 while  $solucin < M$  do
13     Vamos generando elementos aleatorios y los introducimos a la solución
14      $Elementoaintroducir \leftarrow GenerarElementoAleatorio(Elementosrestantes)$ 
15      $Elementosrestantes \leftarrow Elementosrestantes - Elementoaintroducir$ 
16      $solucin \leftarrow solucion \cup Elementoaintroducir$ 
17     Ya tenemos una solución completa y válida de tamaño M
18     El conjunto de elementos restantes solo contiene
19     los elementos que no están en la solución

20  $VectorDistancias \leftarrow GenerarVectorDistancias()$ 
21  $DispersionComparacion \leftarrow CalcularDispersion(VectorDistancias)$ 

22  $Mejora \leftarrow \text{TRUE}$ 
23 while  $Mejora == \text{TRUE} \ \&\& \text{iteraciones} \leq 100000$  do
24     Generamos un vecindario completo de la solución actual
25     y lo mezclamos aleatoriamente
26      $Vecindario \leftarrow GenerarVecindario(solucion)$ 
27      $Vecindario \leftarrow Desordenar(Vecindario)$ 

28     Actualizamos las variables antes de recorrer el vecindario
29      $Copiasolucin \leftarrow solucion$ 
30      $Mejora \leftarrow \text{FALSE}$ 
31      $dispersioncomparacion \leftarrow Dispersion$ 

32     for  $i \in \text{Size}(Vecindario) \ \&\& \text{mejora} == \text{FALSE}$  do
33         Recorremos el vecindario
34          $Copiasolucin \leftarrow SustituirPunto(vecindario[i])$ 
35          $CopiaDistancias \leftarrow GenerarVectorDistancias(Copiasolucion)$ 
36          $dispersioncomparacion \leftarrow CalcularDispersion(CopiaDistancias)$ 

37     if  $dispersioncomparacion < dispersion$  then
38         Si la dispersion es mejor, actualizamos la solución
39          $dispersion \leftarrow dispersioncomparacion$ 
40          $solucin \leftarrow Copiasolucion$ 
41          $Mejora \leftarrow \text{TRUE}$ 
42          $VectorDistancias \leftarrow CopiaDistancias$ 
43          $Restantes \leftarrow CalcularRestantes(solucion)$ 
44     else
45         Si la dispersion no es mejor, no actualizamos la solución,
46         y volvemos al estado anterior
47          $Copiasolucin \leftarrow solucion$ 
48          $CopiaDistancias \leftarrow VectorDistancias$ 
49      $Iteraciones \leftarrow Iteraciones + 1$ 

50 Devolvemos la solución
51 Return  $solucion$ 

```

---



## 4. ESTUDIO EXPERIMENTAL Y ANÁLISIS DE RESULTADOS

En ambos algoritmos hemos usado el mismo vector de semillas, que en cada iteración que ejecuta el programa el algoritmo, se coge la posición i-esima del vector de semillas.

El vector semillas es (1,2,3,4,5) Por lo tanto en la primera iteración se define la semilla como Random::Seed(1), y así sucesivamente.

Para comparar los resultados entre los dos algoritmos implementados en esta práctica, he hecho una tabla donde se muestran, para cada algoritmo, el tiempo medio y la dispersión media conseguida entre las 5 iteraciones conseguido con cada uno de los ficheros de datos.

Algoritmo Greedy				Algoritmo BL			
Caso	Coste medio obtenido	Desv	Tiempo(s)	Caso	Coste medio obtenido	Desv	Tiempo(s)
GAD5-1-125-m2	0.0000	0.00	0.0000000000	GAD5-1-125-m2	0.0000	0.00	0.0000000110
GAD5-2-125-m2	0.0000	0.00	0.0000000172	GAD5-2-125-m2	0.0000	0.00	0.0000001224
GAD5-3-125-m2	0.0000	0.00	0.0000000038	GAD5-3-125-m2	0.0000	0.00	0.0000004575
GAD5-4-125-m2	0.0000	0.00	0.0000000036	GAD5-4-125-m2	0.0000	0.00	0.0000004644
GAD5-5-125-m2	0.0000	0.00	0.0000000036	GAD5-5-125-m2	0.0000	0.00	0.0000001226
GAD5-6-125-m2	30.0224	78.45	0.0001461640	GAD5-6-125-m2	39.2743	67.62	0.0002810220
GAD5-7-125-m2	73.8278	80.87	0.0001471110	GAD5-7-125-m2	74.0016	56.31	0.0001279720
GAD5-8-125-m2	76.3964	81.46	0.0001521120	GAD5-8-125-m2	73.9322	47.63	0.0001100000
GAD5-9-125-m2	83.8874	83.38	0.0001431980	GAD5-9-125-m2	75.0388	40.52	0.0004461210
GAD5-10-125-m2	85.5476	73.80	0.0001571730	GAD5-10-125-m2	77.6711	38.54	0.0001000000
GAD5-11-125-m2	41.3777	95.35	0.0001821140	GAD5-11-125-m2	75.8881	87.88	0.0001311120
GAD5-12-125-m2	41.7288	84.52	0.0001811150	GAD5-12-125-m2	72.7608	83.31	0.0001340220
GAD5-13-125-m2	30.5681	95.22	0.0001811180	GAD5-13-125-m2	75.1791	84.64	0.0001508000
GAD5-14-125-m2	35.8852	95.37	0.0001811170	GAD5-14-125-m2	74.6081	91.06	0.0001190440
GAD5-15-125-m2	75.7863	84.89	0.0002001110	GAD5-15-125-m2	74.6512	79.70	0.0001114220
GAD5-16-125-m2	249.5140	82.90	0.0001890000	GAD5-16-125-m2	119.7030	64.31	0.0002904020
GAD5-17-125-m2	205.4510	76.72	0.0001242030	GAD5-17-125-m2	90.2033	46.67	0.0001800000
GAD5-18-125-m2	117.4670	72.39	0.0001125000	GAD5-18-125-m2	81.2825	52.68	0.0007111700
GAD5-19-125-m2	180.7480	74.52	0.0001790000	GAD5-19-125-m2	113.8439	62.53	0.0011085020
GAD5-20-125-m2	176.8530	73.01	0.0001841000	GAD5-20-125-m2	76.9847	56.79	0.0001895000
GAD5-21-125-m2	87.8713	84.26	0.0001173100	GAD5-21-125-m2	34.4393	59.84	0.0003419320
GAD5-22-125-m2	88.5641	84.57	0.0001701000	GAD5-22-125-m2	17.0638	63.13	0.0003214000
GAD5-23-125-m2	76.1333	84.64	0.0001189000	GAD5-23-125-m2	42.2408	63.67	0.0007110500
GAD5-24-125-m2	73.3271	87.89	0.0001811700	GAD5-24-125-m2	35.5127	71.79	0.0002216500
GAD5-25-125-m2	82.4390	79.34	0.0001170000	GAD5-25-125-m2	36.1338	52.44	0.0007091000
GAD5-26-125-m2	506.5600	66.69	0.0011153000	GAD5-26-125-m2	134.4000	49.71	0.0007212000
GAD5-27-125-m2	471.1100	73.02	0.0001190000	GAD5-27-125-m2	58.5400	64.65	0.0007210000
GAD5-28-125-m2	461.1330	77.89	0.0001900000	GAD5-28-125-m2	103.7200	70.37	0.0007210000
GAD5-29-125-m2	341.3560	58.73	0.0009090000	GAD5-29-125-m2	265.5010	48.21	0.0009100000
GAD5-30-125-m2	461.1360	71.24	0.0009060000	GAD5-30-125-m2	233.1400	64.54	0.0007205000
GAD5-31-125-m2	117.6180	89.96	0.0001813600	GAD5-31-125-m2	39.5049	70.17	0.0004095020
GAD5-32-125-m2	73.7481	74.52	0.0001400000	GAD5-32-125-m2	44.7484	66.37	0.0001640000
GAD5-33-125-m2	138.7900	86.45	0.0001260000	GAD5-33-125-m2	53.6155	68.95	0.0005600000
GAD5-34-125-m2	95.4028	80.39	0.0001510000	GAD5-34-125-m2	42.2846	53.91	0.0004818700
GAD5-35-125-m2	97.1061	83.35	0.0001230000	GAD5-35-125-m2	48.8118	62.37	0.0004100000
GAD5-36-125-m2	471.4890	67.02	0.0003000000	GAD5-36-125-m2	133.7270	53.42	0.0043100000
GAD5-37-125-m2	502.6070	60.63	0.0007100000	GAD5-37-125-m2	122.0150	54.46	0.0046600000
GAD5-38-125-m2	536.6130	64.63	0.0002400000	GAD5-38-125-m2	144.1000	60.46	0.0018100000
GAD5-39-125-m2	510.3990	66.57	0.0021400000	GAD5-39-125-m2	261.1280	53.32	0.0048900000
GAD5-40-125-m2	617.1790	67.18	0.0006700000	GAD5-40-125-m2	403.3000	57.53	0.0004000000
GAD5-41-125-m2	120.1290	80.57	0.0004878100	GAD5-41-125-m2	76.9109	69.66	0.0007100000
GAD5-42-125-m2	114.4890	82.68	0.0001090000	GAD5-42-125-m2	76.6200	65.05	0.0009000000
GAD5-43-125-m2	527.4900	86.06	0.0001640100	GAD5-43-125-m2	81.1052	67.41	0.0011100000
GAD5-44-125-m2	145.4070	82.16	0.0004990000	GAD5-44-125-m2	45.5018	62.68	0.0011089000
GAD5-45-125-m2	124.5110	77.69	0.0001331000	GAD5-45-125-m2	53.9745	69.11	0.0010000000
GAD5-46-125-m2	637.5070	64.29	0.0017120000	GAD5-46-125-m2	143.2090	58.07	0.0007200000
GAD5-47-125-m2	124.0290	56.38	0.0007770000	GAD5-47-125-m2	111.2790	46.99	0.0009100000
GAD5-48-125-m2	461.4800	53.10	0.0001861000	GAD5-48-125-m2	103.8430	54.77	0.0004700000
GAD5-49-125-m2	633.2360	65.34	0.0017010000	GAD5-49-125-m2	137.4790	57.87	0.0013070000
GAD5-50-125-m2	721.4570	65.78	0.0041331000	GAD5-50-125-m2	143.4000	55.83	0.0004800000

(a) Tabla de resultados de Greedy (b) Tabla de resultados de BL

Figura 4: Tablas de resultados de Greedy y BL

<b>Media Desv:</b>	<b>76,5595103744</b>	<b>Media Desv:</b>	<b>55,10878940233</b>
<b>Media Tiempo:</b>	<b>0,008761569604</b>	<b>Media Tiempo:</b>	<b>0,017145496976</b>

(a) Desviación y tiempo de Greedy (b) Desviación y tiempo de BL

Figura 5: Desviaciones y tiempos de Greedy y BL

Observando los datos de las tablas, podemos observar que el algoritmo greedy tiene un tiempo menor que búsqueda local, mientras que tiene una mayor desviación, lo que quiere decir que sus resultados de dispersiones son peores.

Aunque estas diferencias no sean muy significativas, a la hora de evaluar muchas ejecuciones de estos algoritmos, encontramos como se acentúa más la diferencia.

$$\text{Desviacion} = 100 * \sum_{i=1}^n \frac{\text{ValorAlgoritmo}_i - \text{MejorValor}_i}{\text{ValorAlgoritmo}_i} \quad (4)$$

Por lo tanto, tenemos unos datos de referencia, que contienen el mejor coste obtenido para cada instancia del problema. El algoritmo de Búsqueda Local obtiene mejores dispersiones de media que Greedy, y esto es gracias a que este algoritmo tiene mas probabilidad de encontrar mejores soluciones.

Al generar el vecindario completo se asegura que si no se encuentran mejores dispersiones, no las selecciona, al contrario que Greedy, que aunque ninguno mejore la dispersion añade a la solución el que menos la empeore.

Esto evita que el algoritmo de Búsqueda Local vaya hacia soluciones peores(mínimos locales), y siempre se asegure que cuando actualiza la solución es para una mejor dispersion.

En cambio, Greedy acepta soluciones peores a la actual, y esto puede hacer que caiga en mínimos locales, y al siempre añadir elementos a la solución, no poder salir de ellos.

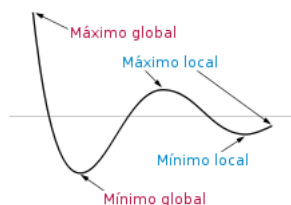


Figura 6: Gráfica que muestra el comportamiento de una búsqueda de una solución

#### 4.1. Tabla resumen

Algoritmo	Desviación media	Tiempo (en segundos)
<i>Greedy</i>	76,5595103744	0,008761569604
<i>BL</i>	55,10878940233	0,017145496976
<i>AGG-Uniforme</i>	40,0088991109	7,463298200000
<i>AGG-Posición</i>	45,4862646141	3,312911800000
<i>AGE-Uniforme</i>	55,9744088626	9,524131200000
<i>AGE-Posición</i>	54,5438597140	5,062808200000
<i>ES</i>	39,6364925123	0,019224743220
<i>BMB</i>	35,4212565508	0,239388720600
<i>ILS</i>	38,7884380920	0,100818801540
<i>ILS_ES</i>	39,6364925123	0,019224843220
<i>BBO</i>	42,0759918345	0,455122914000
<i>BBO-BL</i>	20,5689749201	0,324748746000
<i>BBO-BL+</i>	10,3153847989	8,397473508000

Tabla 1: Tabla de medias de desviaciones y tiempos de los algoritmos

## 4.2. Analisis resultados BBO

A continuacion voy a analizar los resultados de BBO, con todas las variantes realizadas en el

### 4.2.1. BBO sin hibridacion

En la primera prueba del algoritmo hemos visto que nos da dispersion 42, con un tiempo bastante bueno y con los siguientes parametros:

- Tamaño de poblacion : 100
- Numero de iteraciones : 200
- Probabilidad de mutacion : 0.1
- Numero de elites : 20

Observamos que tenemos parametros bastante generosos, lo que va a permitir bastante la exploracion, ya que como hemos comentado anteriormente el BBO en nuestro problema de optimizacion, no tiene una gran calidad de explotacion debido a la propia naturaleza de la funcion objetivo, por lo que hemos decidido probar a explorar de una forma mas exhaustiva, es decir, con una poblacion de tamaño mayor y mayor probabilidad de mutacion.

### 4.2.2. BBO con hibridacion BL

En esta hibridacion con la Busqueda Local hemos usado los siguientes parametros:

- Tamaño de poblacion : 50
- Numero de iteraciones : 100
- Probabilidad de mutacion : 0.01
- Numero de elites : 5

Observamos una muy buena desviacion obtenida de media y un mejor tiempo que en el caso anterior, este se debe a que hemos disminuido bastante el numero de iteraciones(numero de generaciones maximo que se van a generar) y el tamaño de la poblacion.

Aprovechamos mejor la alta exploracion que presenta el BBO, para posteriormente aplicar una BL sobre la poblacion final.

Solo aplicamos dos BL, antes de empezar el BBO y al finalizar.

### 4.2.3. BBO con hibridacion BL++

En esta hibridacion con la Busqueda Local hemos usado los siguientes parametros:

- Tamaño de poblacion : 100

- Numero de iteraciones : 50
- Probabilidad de mutacion : 0.1
- Numero de elites : 5

Observamos la mejor dispersion media hasta el momento y esto se debe a lo que comentamos anteriormente, de que aprovechamos el BBO para una exploracion exhaustiva y aqui aplicamos BBO a todos los elementos de la poblacion en cada iteracion. Es por esto que obtenemos los mejores resultados por el momento.

#### 4.2.4. BBO con hibridacion ES

En esta hibridacion con el Enfriamiento Simulado hemos usado los siguientes parametros:

- Tamaño de poblacion : 100
- Numero de iteraciones : 50
- Probabilidad de mutacion : 0.1
- Numero de elites : 5

Observamos que la hibridacion con Enfriamiento Simulado no ofrece malos resultados, pero bastante peores que las hibridaciones con BL. Esto se puede deber a que el algoritmo de Enfriamiento Simulado con 10000 evaluaciones de la funcion objetivo(que es el ES usado) no llega a explotar bien las soluciones, entonces estamos hibridando un algoritmo que explora mucho como es el BBO, y otro que en nuestro caso tambien esta explorando mucho, por lo que no se puede conseguir una buena explotacion final y es por esto que los resultados no son tan buenos.